DR. ÚRY LÁSZLÓ

COMMODORE PLUS 4 * C 16 * C 116

BASIC ÉS FELHASZNÁLÓI KÉZIKÖNYV

LSI ALKALMAZÁSTECHNIKAI
TANÁCSADÓ SZOLGÁLAT



COMMODORE

PLUS 4 * C 16 * C 116

BASIC ÉS FELHASZNÁLÓI KÉZIKÖNYV

Írta : Dr. Úry László
Dr. Paál Éva C116 fejezet
Tóthné Máriássy Éva C PLUS4 fejezet

Lektorálta : Jarabek Lajos

ISI

ALKALMAZÁSTECHNIKAI TANÁCSADÓ SZOLGÁLAT
BUDAPEST, 1988

E l ő s z ó

Ez a könyv eredetileg csak a COMMODORE 16 személyi számítógép felhasználói kézikönyvének készült. A könyv kiadásának előkészítését az indokolta, hogy a C-16 hazánkban is kezdett elterjedni, elsősorban az iskolákban. Ezért mindenképp szükségét láttuk annak, hogy erre a géptípusra is kiadjunk egy - az előző COMMODORE kiadványokhoz hasonló - összefoglaló jellegű könyvet. A munka már majdnem a végéhez közeledett, amikor kiderült, hogy szép számmal érkeznek hazánkba C-116 és PLUS/4 típusú gépek is. A géptípusok majdnem teljes kompatibilitása miatt úgy döntöttünk, hogy nem adunk ki külön-külön gépkönyveket, hanem e kiadvány végén egy-egy különálló fejezetben ismertetjük e három számítógép egymástól eltérő sajátosságait.

dr. Úry László

Tartalomjegyzék

6.fejezet
Grafikus lehetőségek. Hanggenerálás

7.fejezet
BASIC programozási példák

8.fejezet

Függelék

K Különbségek

1. fejezet

BEVEZETÉS

1.1 A kézikönyv feladata

A Commodore 64 háztartási személyi számítógép sikere arra ösztönözte a cég fejlesztőit, hogy újabb és újabb, a C-64-hez hasonló géptípusokat konstruáljanak. Egyik ilyen - a kimondottan oktatási célokra kifejlesztett Commodore 16 háztartási személyi számítógép. Bár a két számítógép programozása, felépítése igen sok hasonlóságot mutat, lényeges különbségek is vannak köztük. Úgy tűnik, hogy a C-16 kiküszöböli az eddigi Commodore gépek BASIC-jének gyengeségeit. A C-16 video chipje lényegesen jobb, mint a VC-20-é, de nem éri el a C-64-ét. Kihagyták a gépből a beépitett RS-232-es csatornakezelést, aminek pl. a VC-20 esetén semmi értelme sem volt. Elsősorban a felhasználható szabad memóriaterület nagysága - kb. 12 Kbyte - kevés. A C-64 számítógépet adatfeldolgozásra, folyamatirányításra, mérési adatok gyűjtésére is használhatjuk, míg a C-16-ot csak tanulásra, szórakozásra. A C-16 teljesitménye nagyobb, mint a ZX-81, ABC-80, HT 1080Z mikroszámítógépeké. Maximális memória-kiépítés esetén megegyezik a SINCLAIR Spectrum-éval.

A könyv a C-16 programozásához szükséges valamennyi ismeretet tartalmazza. Nem tankönyv azonban, még akkor sem, ha sok és főleg egyszerű BASIC programrészt iktattunk a szövegbe. A könyv feladata kettős: egyrészt **kézikönyvként** a C-16-tal végzett munka segítését végzi, másrészt **útmutatást** kíván adni azoknak, akik már valamilyen mikrogépes és BASIC előismeretek birtokában most kezdenek el a C-16-tal foglalkozni. Elsősorban számukra készült a 7. fejezet, amelyik *csak* programozási példákat tartalmaz, biztosítva a BASIC utasítások szisztematikus megismerését.

1.2 Hogyan használjuk a könyvet?

A könyv fejezetei többé-kevésbé függetlenek egymástól. A géppel még csak most ismerkedőknek először az első három fejezet átolvasását ajánljuk, ezek azok, amelyek elsősorban a BASIC-kel foglalkoznak. További fejezetek és paragrafusok vannak, amelyek

különösebb előismeretek nélkül is megérthetők, ilyenek elsősorban a perifériás egységekkel foglalkozó részek.

Az **1. fejezet** bevezető jellegű, ezen túlmenően csak a C-16 üzembe állítását tartalmazza. A **2. fejezet** a C-16 képernyő szerkesztőjének leírását tartalmazza. A Commodore gépek képernyő szerkesztőinek használata eltér a hasonló nagyságú gépeken megszokottaktól. A másfajta gépekhez szokott olvasó ezért alaposan tanulmányozza át ezt a részt, különben kínos meglelpetések érhetik! A C-64, vagy a VC-20 programozásában jártasak számára egyedül az ESC billentyű használata újdonság.

A **3. fejezet** elején összefoglaljuk a C-16 BASIC interpreter jellemzőit. Ez alsősorban a BASIC programozásban jártas olvasók számára készült. A 3.2 paragrafus a BASIC általános jellemzőit foglalja össze, míg a 3.3 paragrafus a BASIC alapszavakat ismerteti ABC sorrendben.

A **4. és 5. fejezetek** a perifériák és a lemezegység használatát ismertetik. Ez a Commodore gépcsalád valamennyi gépén, így a C-64, a VC-20, a C-116, C-128, Commodore 600-as, 700-as, illetve 8000 sorozatán azonos.

A **6. fejezet** a C-16 további lehetőségeit: a kép és hanggenerálást ismerteti. Az idetartozó BASIC utasításokat a 3.3 paragrafusban természetesen összefoglaltuk, de ebben a fejezetben részletesen ismertetjük az ehhez szükséges ismereteket. A **7. fejezet**, mint már említettük, programozási példákat tartalmaz.

A **8. fejezet** a gépi kódú monitor használatát foglalja röviden össze.

A könyv olvasásához a programozásban való különösebb jártasság nem szükséges. Az alapvető fogalmakat a megfelelő fejezetekben összefoglaljuk. Ezeket az ismereteket egy kezdő már néhány hét alatt minden nehézség nélkül elsajátíthatja. Az I/O-val, illetve a gépi kódu programozással összefüggő részek mélyebb ismereteket igényelnek.

Néhány szót a könyvben használt jelölésekről. A COMMODORE irodalomban a **hexadecimális** (16 alapú) számokat a szám elején levő $ jel vezeti be. A $2C például decimálisan 44 (=2*16+12).

Gyakran van szükségünk az egyes parancs vagy programsorok pontos beírásának az ismeretére. Az ilyen esetekben a vezérlő billentyűk nevét **csúcsos zárójelek közé írjuk.** Abban az esetben, ha a billentyűn két elnevezés is van, akkor csak az egyik nevet írjuk ki. Például a PRINT "<CLEAR>ADATOK" parancsban a <CLEAR>

egyetlen billentyű lenyomását jelenti, a képernyőn pedig egy inverz grafikus jel lesz látható. Abban az esetben, ha a billentyű második jelentése nincs magára a billentyűre ráírva, akkor a csúcsos zárojelen belül a használandó váltó jelét is feltüntetjük, pl. <SHIFT-RETURN>.

A kurzort mozgató billentyűk esetében a nyilak helyett a mozgás irányát jelöljük: <CRSR FEL>, <CRSR LE>, <CRSR JOBBRA> és <CRSR BALRA>. A kurzor fel billentyű nem keverendő össze a <^> (hatványozás jel) billentyűvel!

Vannak olyan esetek, amikor a grafikus jelek maguk is lényegesek, ilyenkor az eredeti programlistát közöljük.

1.3 A C-16 mikroszámítógép üzembe állitása

A C-16 mikroszámítógép hátoldalán és jobb oldalán csatlakozókat találunk. (Lásd a 8. oldalt!). A helyes működés alapfeltétele a kiegészítő berendezések megfelelő csatlakoztatása. A csatlakozókat úgy alakították ki, hogy mindegyikük csak a saját ellendarabjával csatlakoztatható. Röviden ismertetjük a C-16 üzembeállításának legfontosabb lépéseit.

1. A C-16 központi egysége a billentyűzettel egybeépült. A C-16 **transzformátora** külön egység, ezzel van egybe épüitve a hálózati csatlakozó villásdugója. A belőle elágazó kábelt csatlakoztassuk a C-16 jobb oldalán található hálózati csatlakozóba. (Felirata: POWER) A transzformátort magát úgy helyezzük el, hogy mind az - esetleg használt - lemezegységtől, mind a televíziótól a lehető legmesszebb legyen.

2. A transzformátort csatlakoztassuk a 220 V, 50 Hz-es hálózatba.

3. A C-16 számítógép és a televízió összekötésére egyetlen koaxális kábel szolgál. Segítségével kell összekötnünk a C-16-ot a televízióantenna csatlakozójával. Régebbi televíziós készülékek esetén antennaadapterre is szükség lehet. Ezt külön kell beszereznünk. Akiknek a televízióján monitor-bemenet is van, azok ezt közvetlenül is összeköthetik a C-16 monitor csatlakozójával. Az ehhez szükséges kábelt külön kell beszerezni.

4. Helyezzük áram alá a televíziókészüléket, majd a C-16 számítógépet is.

5. Ezután kerülhet sor a televíziós készülék hangolására. Célszerű valamelyik VHF csatornát egyszer s mindenkorra a C-16

frekvenciájára hangolni. A televíziós készüléket úgy kell beállítani, hogy tisztán jelenjen meg az alábbi üzenet:

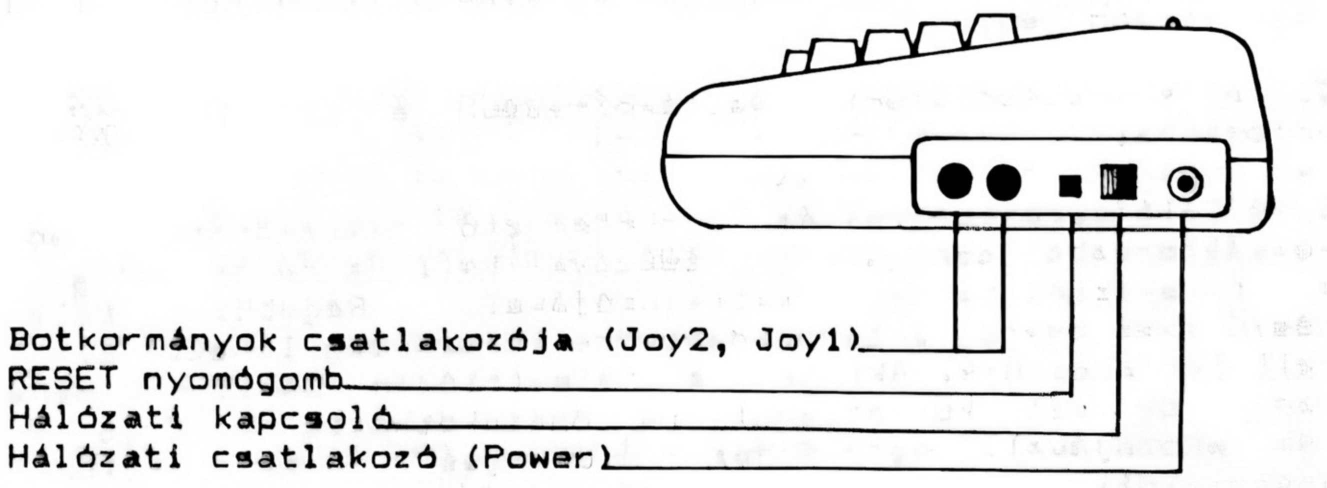
COMMODORE BASIC V3.5 12277 BYTES FREE

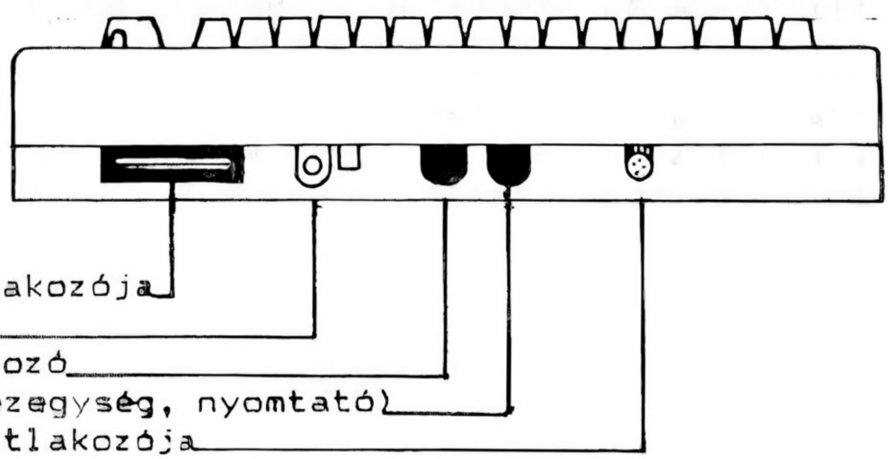
Színes készüléken a feliratnak világoskék keretben, fehér alapon, fekete betükkel kell megjelennie.

A 12277 a BASIC program számára rendelkezésre álló memóriaterület nagyságát jelenti. Ha memóriabővítést használunk (lásd az 1.4 pontot), akkor egy ennél nagyobb szám jelenik meg.

6. Amennyiben további kiegészítő berendezéseket is csatlakoztatunk a C-16-hoz, azt a **C-16 bekapcsolása előtt** kell elvégezni. Ebben az esetben **először** a kiegészítő berendezéseket helyezzük áram alá, s csak azt követően a C-16-ot.

7. Ha a C-16 bővitőjének csatlakozójába kártyát illesztünk, azt **a használt berendezések áram alá helyezése előtt** kell megtennünk. Vigyázzunk, hogy a kártyát megfelelő oldalával felfelé dugjuk be a gépbe!





1.4 Bővítési lehetőségek

A C-16 alapkiépítésében elsősorban az írható/olvasható memória, a RAM kevés. A gépet azonban úgy tervezték meg, hogy nemcsak háttértárakat, hanem közvetlen bővítéseket is illeszthetünk hozzá. (A háttértárakról és egyéb kiegészítő berendezésekről a 4., illetve 5. fejezetekben szólunk részletesen.)

Legfontosabbak a **memória bővítések**. Ezek a bővítőegység csatlakozójába dugható kártyákon kerülnek forgalomba. A kapható bővítések nagysága 16K. A memóribővítő kártyát a számítógép **bekapcsolása előtt** kell a helyére illesztenünk. ügyeljünk arra, hogy a kártya megfelelő oldala legyen felfelé! Helytelen csatlakoztatás esetén a kártya is, a gép is tönkremehet. Memóriabővítés esetén a bejelentkezéskor a gép a használható memóriaterület nagyságát írja ki.

Ugyancsak a bővítőegységhez csatlakoztatandó kártyákon **kész program**okat is vásárolhatunk. Ezek ugyanúgy épülnek fel, mint a memória bővítő kártyák, azzal a különbséggel, hogy a kártyán nem RAM, hanem ROM található. Ez a ROM tartalmazza a beégetett

programot.

A C-16 rendszeres használatához mindenképp célszerű valamilyan háttértárat beszerezni. Felhivjuk a figyelmet arra, hogy a C-16-hoz csatlakoztatható kazettás egységnek éppen úgy **DATASETTE** a neve, mint a C-64-hez csatlakoztathatóé, de típusszáma eltér: **1531.** *A C-64-hez csatlakoztatható botkormányok a C-16-hoz nem jók (és persze fordítva sem).*

A Commodore soros kimenettel rendelkező lemezegységei (VC1541, 4040) közvetlenül csatlakoztathatók a C-16-hoz, s ugyanez áll az ilyen típusu nyomtatókra (MPS801,802,803, SHEIKOSHA stb).

2. fejezet

C-16 alapismeretek

Bevezetés

Ez a fejezet a C-16 személyi számítógép legegyszerűbb felhasználási lehetőségeit, illetve a számítógéppel való kommunikációt ismerteti. A paragrafus olvasása közben célszerű a példákat azonnal ki is próbálni, ezért a következő konfiguráció használatát javasoljuk:

a/ C-16 központi egység,
b/ televízió (vagy monitor).

2.1 A C-16 mint írógép

A képernyő szerkesztő

A C-16 személyi számítógép egy **teljes képernyős szerkesztő**vel rendelkezik. Ez azt jelenti, hogy parancsok bevitelére a képernyő tetszőleges részét használhatjuk, innen az elnevezés. A bevezetésben felsorolt mikroszámítógépek közül egyiknek sem ilyen a szerkesztője. A Sinclear gépek a képernyő utolsó két sorát használják adatbevitelre, az ABC 80 , illetve a HT 1080Z gépeknek egysoros szerkesztőjük van.

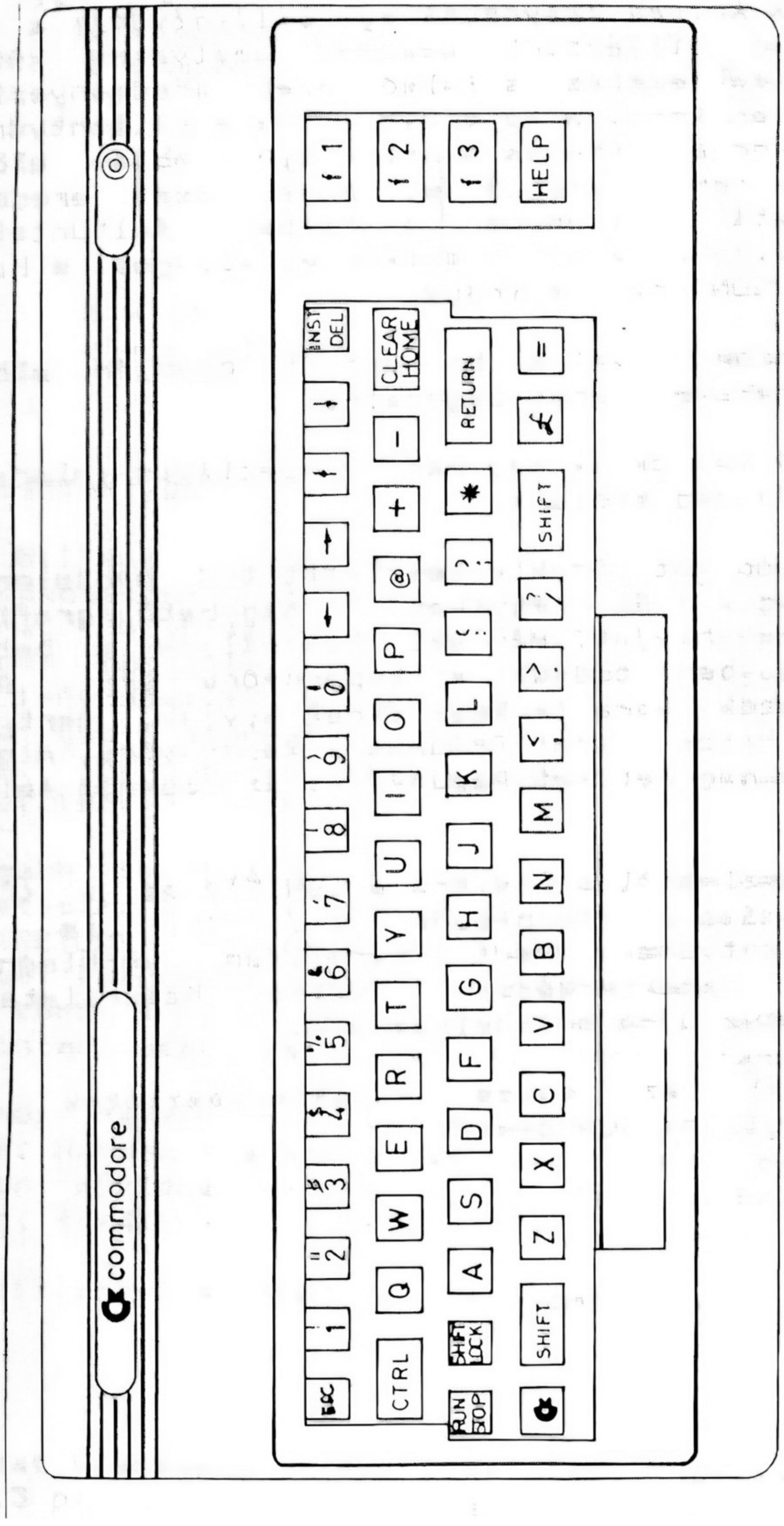
A billentyűzet felosztása

A C-16 billentyűzetén összesen 66 billentyű található. Ezek közül 65 a hardver szempontjából teljesen egyenértékű, csak az interpreter különbözteti meg őket. A baloldali <SHIFT> billentyű fölött egy <SHIFT LOCK> feliratú billentyű található. Ennek lenyomása blokkolja a <SHIFT> billentyűt, azaz ezt követően bármely billentyűt is nyomunk meg, az olyan, mintha a <SHIFT>-et is lenyomva tartottuk volna. A <SHIFT LOCK> újbóli megnyomása feloldja a zárt.

A billentyűzeten a négy úgynevezett **funkció** billentyű jobb oldalt elkülönítve található meg (**f1/f4**,**f2/f5**,**f3/f6**, illetve **HELP/f7** jelöléssel), a többiek egy tömbben, az amerikai írógépszabványnak megfelelően helyezkednek el. Alul egyetlen hosszú billentyű található, ez a **szóköz** (space) billentyű. Ennek a résznek a jobb, illetve bal oldalán helyezkednek el a speciális hatású **vezérlő** billentyűk.

Valamennyi billentyű ismétli önmagát. Ez azt jelenti, hogy ha lenyomva tartjuk a billentyűt, akkor a neki megfelelő karakter folyamatosan a képernyőre kerül. Ez eltér a Commodore régebbi géptípusaitól, amelyeken csak a kurzor vezérlő billentyűk és a <szóköz> ismétli önmagát.

A C-16 billentyűzete



Váltók

Az információ (szövegek, betűk, számok) begépelésére - írógépen használt terminológiával élve - három **váltó** is szolgál. Váltók használata nélkül az a karakter kerül a képernyőre, amelyik a billentyűn látható; vagy amennyiben két jel van a billentyűn, akkor azok közül az alsó.

A <SHIFT> váltót használva (***ami azt jelenti, hogy a*** *<SHIFT> lenyomva tartása közben lenyomunk egy billentyűt!*) a következők történnek. Olyan billentyűk esetén, amelyekre két jel van ráírva, a siftelés (emelés) a felső jelet eredményezi, a neki megfelelő karakter kerül a képernyőre. Ha a billentyűn egyetlen jel látható, akkor a siftelés a billentyű gombján **elöl látható jobb oldali** karakter (grafikus jel) begépelését eredményezi. A váltók használatát bizonyos esetekben feltüntetjük, pl. <CTRL-RVSON>. Olyankor azonban amikor ez világos, elhagyjuk. Az előző példát <RVSON>-nak is írjuk.

A <C=> váltó használatával a billentyűk gombján **elöl látható baloldali** karaktereket lehet begépelni.

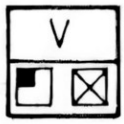
A <CTRL> és <C=> váltóknak még néhány speciális feladata is van, amelyekről később még szólunk.

A C-16 a képernyőn két karakterkészletet tud megjeleníteni (de nem egyidejüleg). Az egyiket '**nagybetűk/grafikus jelek**' karakterkészletnek hívjuk. Ha ezt használjuk a betűket csak nagybetűs alakjukban tudjuk a képernyőre írni. A másikat '**kisbetük/nagybetűk**' karakterkészletnek hívjuk, mert ebben az esetben a betűk kisbetűkként íródnak a képernyőre, míg a <SHIFT> váltót használva nagybetűket kapunk (ez az üzemmód felel meg az *írógép*nek).

Egyik karakterkészletről a másikra a <SHIFT> **és** a <C=> váltók **egyidejű** megnyomásával térhetünk át. (Ez is tekinthető terminátor billentyűnek, mert bármilyen körülmények közt végrehajtja.) A karakterkészlet váltó használata azonban letiltható, illetve újra engedélyezhető.

A váltók hatását az egyes karakterkészletek esetén a következőkben foglalhatjuk össze:

nagy betűk/grafikus jelek	kis betűk/nagy betűk



nagy betűk/grafikus jelek	kis betűk/nagy betűk
V = <V>	v = <V>
╳ = <SHIFT-V>	V = <SHIFT-V>
▌ = <C=-V>	▌ = <C=-V>



nagy betűk/grafikus jelek	kis betűk/nagy betűk
3 = <3>	3 = <3>
# = <SHIFT-3>	# = <SHIFT-3>

Terminátor billentyűk

A legtöbb billentyű lenyomásának nincs közvetlen hatása a számítógép működésére. Hatásukra csak további karakterek íródnak a képernyőre, de számítás, adatátvitel, programsor írása valójában nem történik. Vannak azonban olyan billentyűk, amelyeknek lenyomásával kikerülünk a szerkesztő felügyelete alól, és a gép azonnal elkezd egy konkrét parancsot végrehajtani. Az ilyen billentyűket összefoglaló néven **terminátor** billentyűknek hívjuk.

A legfontosabb terminátor billentyűről, a **<RETURN>**-ről már szóltunk. Hatására a szerkesztő átadja feldolgozásra azt a sort, amelyben a <RETURN> megnyomásának pillanatában a kurzor volt. Ha ez számjeggyel kezdődött, akkor programsor lesz, és bekerül a memóriába a többi programsor közé. Ha nem, akkor parancsnak tekinti az interpreter, és megkísérli végrehajtani.

A <RETURN>-höz nagyon hasonló a funkciója a **<SHIFT-RETURN>** billentyűnek. Hatására a sor írása abbamarad, és a kurzor a következő sor elejére kerül. A beírt sor feldolgozása nem kezdődik meg, *továbbra is szerkesztő módban* dolgozik a gép.

Terminátor billentyű a **<RUN>** is. Megnyomása ekvivalens a

```
DLOAD "*"
RUN
```

parancssorozat kiadásával. (Az utasítások hatásának részletes leírása a 3.3 paragrafusban található meg.) Ennek megfelelően a

<RUN> billentyű megnyomása után az interpeter a lemezegység 0-ás meghajtójában levő első programot betölti a memóriába, majd elindítja. Ha tévedésből nyomtuk meg, akkor a <STOP> billentyű lenyomásával tudjuk a program töltését megszakítani. **Ne keverjük össze** a <RUN> billentyűt és a RUN parancsot. Az előbbi hatására a fentebb vázoltak történnek, míg a RUN parancsot betűnként kell begépelnünk, és utána még a <RETURN> billentyűt is meg kell nyomnunk!

Nem tekinthető igazi terminátor billentyűnek, de itt szólunk a **<STOP>** billentyűről. Hatása csak akkor van, ha egy BASIC program futása közben nyomjuk meg. Ekkor **megállítja** a futó programot. A program ezek után a CONT paranccsal tovább indítható. (Lásd a 3. fejezetben a STOP parancsot!) Ha szerkesztő üzemmódban nyomjuk le, semmilyen hatása sincs.

Az utolsó terminátor 'billentyű' valójában két billentyű egymás utáni lenyomását jelenti. Az első mindig az <ESC> feliratú '**escape**' billentyű. Ez után bizonyos billentyűk megnyomása speciális vezérlő funkciót tölt be. (Részletes ismertetésükre a 20. oldalon kerül sor.)

Hasonlóan itt szólunk a **RESET nyomógombról,** ami a készülék jobb oldalán található. Benyomása a készülék ki-, majd újbóli bekapcsolásával egyenértékű. Használata tehát törli a gépben esetleg tárolt programot!

Vezérlő karakterek

Bizonyos billentyűk megnyomásának nem az a hatása, hogy valamely karakter megjelenik a képernyőn. Ilyenek például a kurzor mozgató karakterek. Ezeket a karaktereket összefoglaló néven **vezérlő karakter**eknek hívjuk. A következőkben ezekről lesz szó.

Színvezérlés

A C-16 számítógép 16 féle alapszínt képes előállítani. Minden egyes színnek - kivéve a feketét - 8 féle árnyalata lehet. Ennek megfelelően a C-16 összesen 8*15+1 = 121 színt jeleníthet meg a képernyőn. A képernyőre kerülő karakterek színének beállítására a <CTRL> vagy a <C=> váltók lenyomva tartása közben egy színbillentyű leütése szolgál. Az ezt követően beirt valamennyi karakter a megfelelő színnel jelenik meg a képernyőn. (A billentyűzés után a kurzor színe *azonnal* megváltozik!) A színbillentyűk azonosak az <1>-<8> számbillentyűkkel. A szín az oldalukra van írva. A felső színek a <CTRL>, az alsó színek a <C=> váltónak felelnek meg.

A színvezérlés fenti változata mindig maximális intenzitással (a

legvilágosabb árnyalattal) kapcsolja be a szóban forgó színt. Ha ettől eltérő intenzitással akarunk a képernyőre irni, akkor a

COLOR 1,<színkód>,<intenzitás>

BASIC utasitást kell használnunk. Ennek hatására a kurzor színe a <színkód>-nak és az <intenzitás>-nak megfelelő lesz. Például COLOR 1,2,4 szürkére változtatja a kurzor színét. Ezt követően valamennyi beírt betű ilyen színü lesz.

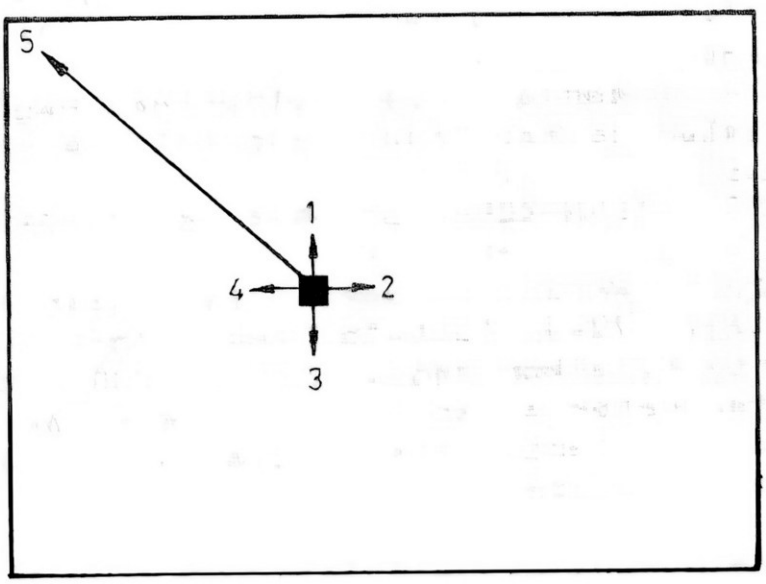
Kurzor vezérlés

A kurzor billentyűk teszik lehetővé, hogy a kurzort - a képernyőn látható információ megváltoztatása nélkül - tetszőleges helyre pozicionáljuk. Hat olyan billentyű van, amelyeket ebbe a kategóriába sorolunk.

A **<CLEAR>** billentyű lenyomása **törli a képernyőt**, és a kurzort a bal felső sarokba viszi vissza. (Ez az ún. **home** pozició.) A **<HOME>** lenyomása nem törli a képernyőt, csak a 'home' pozicióba viszi a kurzort.

A **<CRSR JOBBRA>**, a **<CRSR BALRA>**, a **<CRSR FEL>** és a **<CRSR LE>** billentyűk a rajtuk levő nyilak irányának megfelelően mozgatják a kurzort, a képernyő tartalmának megváltoztatása nélkül. Abban az esetben, ha a képernyő alsó sorában nyomjuk meg a <CRSR LE> billentyűt, a szerkesztő a teljes képernyőt feljebb tolja és így alul egy üres sor keletkezik. A képernyő első sorában a <CRSR FEL>-nek nincs hasonló hatása, egyszerűen hatástalan. Ha a képernyő jobb vagy bal szélén lépünk át a <CRSR JOBBRA> vagy a <CRSR BALRA> billentyűvel, akkor a képernyő másik szélén, egy sorral lejjebb illetve feljebb lépünk vissza a másik oldalon.

1 = <CRSR FEL>
2 = <CRSR JOBBRA>
3 = <CRSR LE>
4 = <CRSR BALRA>
5 = <HOME>



Logikai sor

Mint már említettük, s mint ahogy ez azonnal látszik, a képernyőn egy sorba negyven karaktert lehet írni. Ezt a sort **fizikai** sornak hívjuk, mert a képernyő valódi méretét jelenti. Negyven karakter azonban meglehetősen kevés. A Commodore gépekhez vásárolható legtöbb nyomtató például 80 karakteres sorokat kezel. éppen ezért a C-16 szövegszerkesztőjét úgy tervezték meg, hogy képes **több, egymás alatti** sort egyetlen **logikai sor**nak tekinteni. Két fizikai sorból akkor lesz egyetlen logikai sor, ha a képernyő jobb szélén - a kurzor vezérlő billentyű kivételével - valamilyen billentyűvel átirunk. Ilyen módon pl. - figyelembe véve a puffer hosszát is - a BASIC programsorok hossza maximum 88 karakter lehet.

A képernyőn nem látszik, hogy mely egymás alatti sorok lettek logikai sorrá összekapcsolva. Ebből néha kellemetlenségek adódnak.

Az interpreter a logikai sor fizikai sorait a következő esetekben kezeli ***egynek***:

a/ karakterek törlése/beszúrása;
b/ a képernyő fel/le történő görgetése;
c/ a sor feldolgozásra való átadása.

Az a/ esetben, ha a logikai sor első fizikai sorában törlünk egy karaktert, akkor a második fizikai sor is egy hellyel balra tolódik. Hasonlóan, ha beszúrunk egy karaktert, a második sor is egy hellyel jobbra tolódik.

Bizonyos esetekben (például ha a képernyő utolsó során túl írunk) a képernyő egy sorral felfelé vagy lefelé tolódik el. Sor alatt itt mindig **logikai** sor értendő. Ha a képernyőt felfelé kell eltolni, és az első két sor egyetlen logikai sort alkot, akkor fizikailag a képernyő két sorral tolódik el felfelé.

Az interpreter a c/-ben jelzettnek megfelelően a teljes logikai sort feldolgozza. Irjuk be például a következőket:

A<SHIFT-RETURN><CRSR BALRA><SZóKöZ>?2+2

A kurzort tartalmazó sorban látszólag egy szintaktikusan helyes utasitás: ?2+2 látható. Ha most megnyomjuk a <RETURN> billentyűt, akkor mégis ?SYNTAX ERROR hibajelzést kapunk, mert az interpreter a sort **A?2+2**-ként értelmezi. A <CRSR BALRA> <Szóköz> billentyűzéssel ugyanis a két sor egyetlen logikai sorrá állt össze.

Inverz karakterek

Normális esetben a képernyőre kerülő betűk pontjai lesznek a háttértől eltérő színűek. Lehetőség van azonban arra is, hogy a karakterhely maradék pontjai legyenek a betű színével megvilágítva, míg maga **a betű háttérszínű** maradjon. Úgy is mondhatnánk, hogy a betű negatívját írjuk ki. A <CTRL-RVSON> billentyű lenyomását követően valamennyi karakter inverz alakban íródik ki. Normál üzemmódra a <CTRL-RVSOFF> billentyű leütésével térhetünk vissza. A fenti két billentyű leütésekor a képernyőn - látszólag - semmi sem történik. Ha azonban egy további karaktert leütünk, a hatás azonnal jelentkezik. Próbáljuk ki a

<CTRL RVSON> A <CTRL RVSOFF> A

beírását!

Az előzőekben említettük, hogy a <RETURN> billentyű megnyomása után a képernyő szerkesztő azt feltételezi, hogy a kurzort tartalmazó sort kívántuk feldolgozásra átadni az interpreternek. (Vannak olyan mikrogépek, ahol adatok és programsorok bevitelére csak a képernyő utolsó sorát használhatjuk) Emiatt a tulajdonsága miatt hívjuk a C-16 szerkesztőjét *teljes képernyős* szerkesztőnek. Ez lehetővé teszi, hogy a képernyőn levő szövegrészeket felhasználjuk a parancs vagy programsor végleges "megfogalmazásához".

Különösen kényelmes ez hibás programsorok javításánál. Kilistázzuk a szóban forgó programsort, majd a vezérlő karakterek segítségével kijavítjuk. Ha a <RETURN> megnyomásának pillanatában a kurzor - bárhol - a javított sorban van, az interpreter a régi programsort az éppen javítottra cseréli.

Karakterek villogtatása

Lehetőség van a képernyőre kerülő karakterek villogtatására. Az ilyen feliratok kiugranak a többiek közül, azonnal magukra hívják a figyelmet. Ha villogó karaktert akarunk a képernyőre vinni, akkor nyomjuk le a <CTRL FLASHON> billentyűt. Az ezután a képernyőre kerülő karakterek folyamatosan villognak egészen addig, amíCg a képernyőn vannak. Ha már nem akarunk több villogó karaktert a képernyőre írni, akkor csak a <CTRL FLASHOFF> billentyűt kell megnyomnunk. Ezt követően a karakterek újból villogás nélkül kerülnek a képernyőre. Irjuk próbaként be a következőt:

<CTRL FLASHON> A <CTRL FLASHOFF> A

A képernyőre kerülő két 'A' betű közül az első villog, a második nem.

A képernyő felosztása

Első pillantásra is látszik, hogy a képernyő két részre oszlik: egy téglalap alakú, az információt hordozó részre, illetve egy azt körülölelő egyszínű keretre. A középső részben 25 sorba, soronként 40 karakter írható. Speciális módon szabályozható azonban az, hogy ennek a területnek melyik - ugyancsak téglalap alakú - részét használja az interpreter. Ezt a részt hívjuk **ablak**nak. Az ablakok használata lehetővé teszi, hogy például a képernyő felső részét a program eredményeinek kiírására, alsó felét listázásra, illetve a program szerkesztésére használjuk. Lehetőség van arra, hogy a képernyő két felén ugyanannak a programnak két különböző részét listázzuk ki. Az 'ablak' definiálására az <ESC> billentyűt használhatjuk az alább részletezendő módon.

Az <ESC> billentyű

Az <ESC> billentyű megnyomása után bizonyos billentyűket megnyomva további vezérlő funkciókat hajthatunk végre, összesen 18-at. A Commodore 64 és a VC-20 gépek nem rendelkeznek ezekkel a lehetőségekkel, csak a CBM 600-as, illetve 700-as sorozatu gépei. A továbbiakban ezeket a funkciókat ismertetjük.

<ESC> majd <A>: beszúrási üzemmód bekapcsolása
Alapállapotban a kurzor helyén levő karaktert **írja felül** az a jel, amit éppen begépeltünk. Beszúrási üzemmódban a teljes sor - a kurzor helyén levő karaktert is beleértve - egy hellyel jobbra tolódik, s az éppen benyomott billentyűnek megfelelő karakter a kurzor alatt levő, immár üres helyre szúródik be.

<ESC> majd <B>: az *ablak* jobb alsó sarkának beállitása
A kurzor aktuális helye lesz az ablak jobb alsó sarka.

<ESC> majd <C>: törli a beszúrási üzemmódot.

<ESC> majd <D>: sor törlése
A billentyűzés hatására törlődik a kurzort tartalmazó sor. A lejjebb levő sorok eggyel feljebb csúsznak. Alul egy üres sor keletkezik.

<ESC> majd <I>: sor beszúrása
A billentyűzés hatására a kurzort tartalmazó sortól kezdve valamennyi sor eggyel lejjebb csúszik. Az utolsó sor elvész.

<ESC> majd <J>: kurzor a sor elejére
A kurzor annak a sornak az első oszlopába kerül, amelyikben éppen áll.

<ESC> majd <K>: kurzor a sor végére
A kurzor annak a sornak az utolsó nem szóköz helyére kerül, amelyikben áll.

<ESC> majd <L>: képernyő görgetés bekapcsolása
Alapállapotban így dolgozik az interpreter. Ez azt jelenti, hogy ha a képernyő utolsó során **túl** írunk, akkor az egész képernyő tartalma egy sorral feljebb csúszik. Lehetőség van ennek a módnak a megszüntetésére.

<ESC> majd <M>: képernyő görgetés kikapcsolása
Ebben az esetben, ha a képernyő utolsó sora után írunk, a képernyő tartalma nem tolódik egy sorral feljebb, hanem e helyett az írás az **első** sorban folytatódik.

<ESC> majd <N>: normál üzemmód
Visszaállítja a 40*25-ös ablakméretet, és törli a képernyőt.

<ESC> majd <O>: egyszerre kikapcsolja az inverz karakterírási, illetve villogtató funkciókat.

<ESC> majd <P>: sor törlése a kurzorig
A billentyűzés törli a sort egészen a kurzorig. A kurzor helyén álló karakter is elvész. A sor többi karaktere a helyén marad.

<ESC> majd <Q>: sor törlése a sor végéig
A kurzor helyén álló karaktertől a sor végéig valamennyi karakter törlődik.

<ESC> majd <R>: a képernyő kicsinyítése
Beállítja a 38*23-as ablakméretet, s ezzel egyidejüleg törli a képernyőt.

<ESC> majd <T>: ablak bal felső sarkának beállitása
Az ablak bal felső sarka a kurzor jelenlegi helyzete lesz.

<ESC> majd <V>: képernyő felfelé tolása
A képernyő tartalmát egy sorral feljebb tolja. Alul egy üres sor keletkezik.

<ESC> majd <W>: képernyő lefelé tolása
A képernyő tartalmát egy sorral lejjebb tolja. Felül egy üres sor keletkezik.

<ESC> majd <X>: hatástalan
Megnyomása hatástalanítja az <ESC> megnyomását. Ha tehát az <ESC> billentyűt *tévedésből* nyomtuk meg, akkor az <X> billentyű lenyomásával hatástalaníthatjuk.

Ha egyszerre több ablakot is szeretnénk használni, akkor ezt a legegyszerűbben ugy érhetjük el, hogy egy rövid programot

állandóan a memóriában tartunk, s az <f2> illetve <f3> billentyűhöz hozzárendeljük az egyik, illetve a másik ablak beállítását. Egy ilyen program a következő:

```
10 KEY 1,"RUN99"+CHR$(13)
15 KEY 2,"GOSUB30"+CHR$(13)
20 KEY 3,"GOSUB45"+CHR$(13)
25 END
30 ?CHR$(19)+CHR$(19)+CHR$(27)+"T";
35 FOR I=1 TO 11: ? CHR$(17);: NEXT I
40 ?CHR$(157)+CHR$(27)+"B"+CHR$(19);: RETURN
45 ?CHR$(19)+CHR$(19);
50 FOR I=1 TO 11: ? CHR$(17);: NEXT I
55 ?CHR$(27)+"T";
60 FOR I=1 TO 12: ? CHR$(17);: NEXT I
65 ?CHR$(157)+CHR$(17)+CHR$(27)+"B"+CHR$(19);: RETURN
99 REM
```

A programot töltsük be, majd adjuk ki a RUN parancsot. Ettől kezdve - amíg a program a memóriában van - az <f2> lenyomása után a képernyő felső, az <f3> lenyomása után az alsó felét használhatjuk képernyőnek. Ha programot akarunk írni csak a 100-nál nagyobb sorszámokat használhatjuk. Az <f1> billentyű ekkor is a megfelelő helytől indítja el a programot.

A 30.-40. sorokban az ablak bal felső sarkát a *home* pozícióra, a jobb alsó sarkát pedig a 12. sor 39. karakterhelyére állítjuk be. A 30., illetve a 45. sorban töröljük a beállított ablakot. A 45.-65. sorokban az ablak bal felső sarkát a 13. sor 1., míg jobb alsó sarkát a 25. sor 39. karakterhelyére állítjuk be a megfelelő ESC karakterek kinyomtatásával.

Idézőjel üzemmód

Bizonyos feltételek mellett a képernyő-szerkesztő a vezérlő billentyűket szabályszerű üzemmódjuktól eltérően kezeli. Ezek egyike az un. 'idézőjel' üzemmód.

Idézőjel üzemmódban, azaz megnyitott idézőjel után, a 'közönséges' karakterek szabályszerűen kiíródnak a képernyőre, a vezérlő karakterek azonban nem hajtódnak végre. Helyettük inverz karakterek íródnak ki, amelyek valójában a lenyomott vezérlő karaktereket **reprezentálják**. Ez lehetővé teszi, hogy a programokban a sztringek közé kurzor- és színvezérlő karaktereket tegyünk. Ennek köszönhető, hogy amikor az idézőjelben levő szöveg kiíródik a képernyőre, akkor a sztring részeként automatikusan végrehajtódnak a megfelelő vezérlő funkciók. Példaként gépeljük be a következő sort:

PRINT "<CLR><CRSR LE><CRSR LE><CTRL-RVSON>A" <RETURN>

Amíg nem nyomjuk meg a <RETURN> billentyűt, nem történik semmi; a sztringbe egyszerűen grafikus jelek kerülnek. A <RETURN> megnyomása után az interpreter törli a képernyőt, és a képernyő második sorába egy inverz A betűt ír.

Idézőjel üzemmódban a <CTRL-RVSON> sem hajtódik végre, hatására egy inverz R jelenik meg az idézőjelen belül. Az ezután beírt karakterek a sztring kiírása során inverz formában jelennek majd meg. Az inverz kiírás hatásának megszüntetésére a <CTRL-RVSOFF> billentyűt kell leütnünk. (Ennek hatására is egy inverz grafikus jel kerül a sztringbe.) Például:

```
10 PRINT "<CTRL-RVSON>KUTYAFULE<CTRL-RVSOFF>";
20 PRINT "KUTYAFULE"
```

A fenti rövid kis program kétszer írja ki a "KUTYAFULE" sztringet, először inverz formában, utána normálisan.

Idézőjel üzemmódban azok a karakterek is inverz alakú grafikus jelként íródnak a sztringbe, amelyeknek amúgy nincs is hatásuk. Például a <CTRL-A> hatására egy inverz alakú 'a' íródik ki.

A <DEL> billentyű (lásd lejjebb) az egyetlen vezérlő billentyű, amelyre az "idézőjel" üzemmód nincs hatással. Igy, ha egy hibát vétünk idézőjel üzemmódban, a <CRSR BALRA> billentyűt nem használhatjuk visszaléptetésre. E helyett a megkezdett sort a <SHIFT-RETURN> billentyű lenyomásával be kell fejeznünk. Sztringben használható kurzor vezérlő karakterek az alábbiak:

Vezérlő karakter

<CRSR FEL>
<CRSR LE>
<CRSR JOBBRA>
<CRSR BALRA>
<CLEAR>
<HOME>
<INST>

Karakterek törlése/beszúrása

A szerkesztő lehetőséget nyújt arra is, hogy egy-egy sorba karaktereket szúrhassunk be, vagy törölhessünk. Az <INST> billentyű lenyomása a kurzor alatt, illetve attól jobbra levő karaktereket egy hellyel jobbra tolja. Igy a kurzor alatt egy szóköz keletkezik, ahova új karaktert írhatunk be. A szerkesztő megjegyzi, hogy az <INST> billentyűt használtuk, és egész addig,

míg ezeket a helyeket be nem töltöttük ún. 'inzert' vagy *beszúrási üzemmódban* dolgozik. Az <INST> hatását a következő ábrán szemléltethetjük:

```
  a kurzor helye
               |
               v
    ABCDEFGHIKLMNO     <INST>
    ABCDEFGHI KLMNO    <J>
    ABCDEFGHIJKLMNO
```

Beszúrási üzemmódban a kurzor és a színvezérlő karakterek ismét inverz karakterként íródnak ki (úgy mint idézőjel üzemmódban). Az egyetlen különbség az <INST> és <DEL> beszúró illetve törlő billentyű hatásában van. A <DEL> ahelyett, hogy szabályosan működne, most inverz 'T'-ként íródik ki. A <INST> billentyű, amely inverz karaktert hoz létre "idézőjel" üzemmódban, most szabályosan szóközt szúr be.

Ezt a jelenséget felhasználhatjuk olyan PRINT utasítás létrehozására, amelyik törlést (<DEL>) tartalmaz. (Ez idézőjel üzemmódban lehetetlen.) Az inzert üzemmódot a <RETURN>, <SHIFT-RETURN>, <RUN> billentyűk lenyomásával vagy valamennyi hely betöltésével lehet megszüntetni.

Példa <DEL> karakterek használatára sztringben (így is kell billentyűzni!):

```
10 PRINT "HELLO" <DEL> <INST> <INST> <DEL> <DEL> P" <RETURN>
```

Ha ezt a példa-programot a RUN parancs kiadásával lefuttatjuk, a HELP szó fog kiíródni, mert az LO betűk törlődnek, mielőtt a P kiíródna. A sztringben levő törlő karakter hatása a LIST és PRINT utasítás esetén egyaránt érződik. Ezzel a módszerrel el lehet 'rejteni' a sor egy részét vagy a program egy teljes sorát. Az ilyen sorok újraszerkesztése természetesen igen nehézkes.

Az eddig említetteken kívül is van még néhány olyan vezérlő karakter, amely speciális módon helyezhető csak el egy sztringkonstansban. Ahhoz, hogy ezeket beírhassuk, részükre üres helyeket kell hagyni a sztringben, majd lenyomni a <SHIFT-RETURN> billentyűt, és utána újból szerkeszteni a sort. A javitás megkezdése elött nyomjuk le a <CTRL-RVSON> billentyűt. A sztringbe most már gépelhetünk inverz karaktereket is. (Az idézőjelet természetesen nem nyithatjuk meg újra. A szerkesztő nem kerül idézőjel üzemmódba, ha a kurzorral átlépünk egy idézőjelen. Ugyanigy, ha a megnyitott idézőjelet a <DEL> billentyűvel töröljük, a szerkesztő továbbra is idézőjel üzemben dolgozik! Az inverz alakú vezérlő karakterek a következők:

Funkció	**Billentyűzés**
<RETURN>	M
<SHIFT-RETURN>	<SHIFT> M
kisbetűk/nagybetűk	N
nagybetűk/grafikák	<SHIFT> N

A fenntartott helyre gépeljük be a megfelelő inverz karaktert, majd nyomjuk meg a <RETURN> billentyűt. A sztring kiírásakor a megfelelő vezérlő funkció hajtódik végre.

Irjuk be a következő sorokat:

10 REM " FOPROGRAM<SHIFT-RETURN>
<CRSR FEL>,<CRSR JOBBRA> **ez utóbbit nyolcszor!**
<CTRL-RVSON>,<SHIFT-M><RETURN>

A LIST 10 parancs a programsort a következő alakban listázza:

10 REM "
FOPROGRAM

A PRINT utasításban **szereplő** vezérlő karaktereket a most vázolt módszernél sok esetben egyszerűbben lehet CHR$(B) alakban előállitani. Az alábbi táblázat összefoglalja a felhasználható vezérlő karaktereket:

Funkció	**B értéke**
<ESC>	27
<RETURN>	13
<SHIFT-RETURN>	141
kisbetűk/nagybetűk	14
nagybetűk/grafikák	142
<SHIFT-C=> letiltása	8
<SHIFT-C=> engedélyezése	9
Home	19
CLR	147
Kurzor le	17
Kurzor fel	145
Kurzor jobbra	29
Kurzor balra	157
Inverz karakterek	18
Normál karakterek	146
Villogtatás	130
Villogtatás kikapcsolása	132

Törlés	20
Beszúrás	148
f1	133
f3	134
f5	135
f7	136
f2	137
f4	138
f6	139
<HELP>	140
Fehér	5
Piros	28
Zöld	30
Kék	31
Narancssárga	129
Fekete	144
Barna	149
Sárgászöld	150
Rózsaszín	151
Kékeszöld	152
Világoskék	153
Sötétkék	154
Világoszöld	155
Bíbor	156
Sárga	158
Encián	159

2.2 **A C-16 mint kalkulátor**

Legegyszerűbb esetben a C-16-ot az alapműveletek elvégzésére, bizonyos részeredmények tárolására használjuk. (Ezek azok a feladatok, amelyeket egy zsebszámítógéppel is el lehet végezni). Ehhez mindössze két BASIC utasítást kell ismerni, ezek az

a/ értékadó,
b/ nyomtató utasítások.

Az **értékadó utasítások**

<változó> = <kifejezés>

alakúak, ahol a <változó> annak a memóriarésznek a neve,

amelyben a <kifejezés> értékét tárolni szeretnénk. A legegyszerűbb értékadások azok, amelyek a <változó> értékét konstansként adják meg: X=2.53 vagy Y=-15.001.

Ennél egy fokkal bonyolultabb például a Z=X*X+Y*Y értékadás. A változók és kifejezések írásának meghatározott szabályai vannak. Ezek lényegében a matematikában tanult szabályokkal egyeznek meg. Részletesen erről a 3. fejezetben szólunk.

A nyomtató utasítás szerkezete, használata az értékadó utasításénál lényegesen bonyolultabb. Itt csak egyik legegyszerűbb alakját adjuk meg, ami céljainknak teljes egészében megfelel (Bővebben a 3. fejezetben írjuk le használatát.)

PRINT <kifejezés1>, <kifejezés2>,...

A PRINT helyett egyszerűen egy kérdőjelet (?) is írhatunk.

Most már mindent tudunk ahhoz, hogy a C-16-tal bonyolult számításokat is végezhessünk. Nézzük meg, hogyan számíthatjuk ki egy derékszögű háromszög átfogóját (C) a Püthagorász tétel segítségével, ha adott a két befogó (A és B). Legyen mondjuk A=4, B=3. Egy szám négyzetgyökének kiszámítására az SQR függvény szolgál. A következő parancsokat kell végrehajtanunk:

```
A=4 <RETURN>
B=3 <RETURN>
? SQR(A^2 + B^2) <RETURN>
```

Válaszként a képernyőn a következő üzenet jelenik meg:

```
 5

READY.
█ ←—— villog
```

A utolsó sort a következőkkel is helyettesíthetjük:

```
C=SQR(A^2+B^2) <RETURN>
? C <RETURN>
```

Ez utóbbi esetben előbb kiszámítjuk a C értékét, tároljuk és csak utána írjuk ki. A C-16 szövegekkel is képes dolgozni. Tetszőleges karaktersorozatot **sztringnek** hívunk. Azok a változónevek, amelyek sztringeket tárolnak a $ jelre végződnek. A

```
X$="KUTYA":Y$="FULE" <RETURN>
?X$+Y$ <RETURN>
```

parancsok végrehajtása után a képernyőn a *KUTYAFULE* felirat jelenik meg. Az első sorban szereplő **kettőspont (:) két utasítás elválasztására** szolgál, így a <RETURN> billentyűt csak a legvégén kell benyomni. Sztringek körében egyetlen 'hagyományos' művelet van csak, az összeadás, ami a sztringek egymáshoz fűzését jelenti.

Eredeti példánkat így is módosíthatjuk:

```
A=4: B=3: C=SQR(A^2+B^2) <RETURN>
?"BEFOGOK= ";A,B: ?"ATFOGO= ";C <RETURN>
```

Válaszként az alábbiakat kapjuk:

BEFOGOK= 4 3
ATFOGO= 5
READY.

■ ←—— villog

Programok írása

A BASIC interpreter lehetőséget ad arra is, hogy a begépelt utasításokat ne hajtsuk végre azonnal, hanem tároljuk a memóriában, ahol ezek később egyetlen parancs kiadásával végrehajthatók lesznek. Ahhoz, hogy egy parancs programsorrá váljon, nem kell mást tenni, mint eléje egy sorszámot írni. Ha egy sor számmal kezdődik, az mindig programsornak számít.

A derékszögű háromszög példáját programként a következőképpen írhatjuk meg:

```
10 A=4: B=3: C=SQR(A^2+B^2) <RETURN>
20 ?"A BEFOGOK= ";A,B <RETURN>
30 ?"AZ ATFOGO=";C <RETURN>
```

Ezután a RUN parancs kiadása (azaz a RUN <RETURN> beírása) a fenti programot lefuttatja. Ugyanazt az eredményt kapjuk, mintha a megfelelő utasításokat soronként mi magunk végeztük volna el. A programot az <f6> billentyű lenyomásával is elindíthatjuk.

A fenti program csupán a 3, 4 befogójú derékszögű háromszög átfogóját képes kiszámítani. Ahhoz, hogy tetszőleges derékszögű háromszög átfogóját kiszámíthassuk, egy új utasítást kell használni, amivel egy **már futó programnak adatokat adhatunk át.** Módosítsuk a program 10. sorát:

```
10 INPUT A,B: C=SQR(A^2+B^2) <RETURN>
```

A RUN <RETURN> billentyűzése után a képernyő következő sorának elején egy kérdőjel (?) jelenik meg; jelezve, hogy a program adatot vár (ez az INPUT utasítás feladata). Az adatok bebillentyűzése után a megfelelő értékadások végrehajtódnak, és a program ezekkel az értékekkel számol tovább. A

3,4 <RETURN>

válasz után A értéke 3, B értéke 4 lesz. Ilyen módon természetesen tetszőleges derékszögű háromszög átfogóját kiszámíthatjuk.

Vezérlésátadó utasítások

Rendes körülmények közt a BASIC interpreter a memóriájában tárolt programsorokat **növekvő sorrendben hajtja végre.** A fenti program esetén az utasítások végrehajtásának sorrendje: 10, 20, 30. Vannak azonban olyan utasítások, amelyek feladata éppen **az utasítások végrehajtási sorrendjének a megváltoztatása.** Ezek közül sorolunk fel néhányat. Részletes leírásuk a 3. fejezetben található meg.

a/ GOTO <sorszám>
Az utasítás hatására a program futása a <sorszám> számú programsortól folytatódik. Ha több derékszögű háromszög átfogóját is ki akarjuk számítani, akkor a fenti programot a következő sorral egészítsük ki:

40 GOTO 10 <RETURN>

A program futása újra és újra visszatér a program elejére, és így akárhány derékszögű háromszög átfogóját ki tudjuk számítani. A program futását a <STOP> billentyű lenyomásával állíthatjuk csak meg.

b/ **IF** <feltétel>**THEN** GOTO <sorszám>
Az utasítás végrehajtása a GOTO-ban megjelölt sorszámú utasítással folytatódik, ha a <feltétel> igaz. Ha nem, a program a következő sorszámú utasítás végrehajtását kezdi el. Előző programunkat kiegészíthetjük az adatok valódiságát ellenőrző sorral:

15 IF A<=0 OR B<=0 THEN GOTO 10

(Itt <= a *kisebb egyenlő*nek felel meg.) Ha a fenti feltétel teljesül, akkor rossz adatokat adtunk meg, és a program a végeredmény kiírása helyett új adatokat kér.

c/ **FOR** <valós változó>=<kezdőérték> **TO** <végérték>
Ez az utasítás az úgynevezett ciklus utasítás, mert segítségével lehetővé válik bizonyos programrészek megadott számú ismétlése. Ha például előre tudjuk, hogy N darab derékszögű háromszög átfogóját kell kiszámítani, akkor ezt a következő programmal végezhetjük el:

```
10 INPUT "N=";N
20 FOR I=1 TO N
30 INPUT "a,b=";A,B
40 IF A<=0 OR B<=0 THEN GOTO 30
50 C=SQR(A^2 + B^2)
60 PRINT "BEFOGOK="; A;B; "ATFOGO=";C
70 NEXT I
```

(Ne felejtsük el a programsorok illetve a parancsok beírása után a <RETURN> billentyű lenyomását!) A RUN parancs kiadása után először az N értékét kell megadnunk (pl. 3). Ezt követően 3 derékszögű háromszög átfogóját számíthatjuk ki. I értéke a kezdőérték, vagyis 1 lesz, és a program a következő NEXT I utasításig fut. Ezután I értéke megnő eggyel, majd az interpreter ellenőrzi, hogy nem léptük-e túl a 3-at. Ha nem, akkor a 20-70 közti programsorok újból végrehajtódnak. Ha igen, a következő utasítással folytatódik a program. (Ha ilyen történetesen nincs, a program futása véget ér.)

2.3 **BASIC programok szerkesztése és tárolása**

A képernyő szerkesztő használatával már megismerkedtünk. Speciális BASIC parancsok a memóriában tárolt program szerkesztését (módosítását) teszik lehetővé.

A BASIC programsorokat tetszőleges sorrendben gépelhetjük be. Kezdhetjük a legkisebb sorszámú sorral, de akár a legnagyobbal is. Természetesen - a javításokat kivéve - növekvő sorrenben szokás beírni a sorokat. A programok sorszámát ne egyesével növeljük. Ha egy új sort kell beszúrnunk a program javításakor, és a sorok száma egyesével nő, nem tudjuk hova beírni. Növekménynek 5-öt vagy 10-et szokás választani. A C-16 lehetőséget nyújt arra, hogy a RENUMBER parancs segítségével a program sorait újraszámozzuk. Például a RENUMBER 20,5 <RETURN> parancs kiadása után az interpreter a program sorait 20-tól kezdődően 5-tösével növekedve számozza meg.

LIST
A fenti utasítás a képernyőre listázza a memóriában tárolt programot. (Hasonló hatást érünk el az <f7> funkcióbillentyű lenyomásával.) Ha csak a program bizonyos sorszámú soraira vagyunk kíváncsiak, akkor meg kell adnunk az első, illetve utolsó listázandó sort. Például

LIST 127

a 127. sorszámú sort listázza ki.

A kilistázott programsorokat a képernyő szerkesztő vezérlő karaktereivel (<CRSR>, <DEL> stb.) szerkeszthetjük. A kívánt módosítások után a <RETURN> megnyomása a memóriában tárolt programsort a képernyőn láthatóra cseréli.

A program egy adott sorát egy ugyanolyan sorszámú üres sor bevitelével törölhetjük. Például a 15 <RETURN> hatására törlődik a 15.sor (ha volt ilyen sorszámú sor). Több programsor egyidejű törlésére a **DELETE** parancs szolgál. Például a

DELETE 120-450

parancs a 120. sortól kezdve, a 450. sorral bezáróan valamennyi sort törli.

NEW
A parancs hatására a teljes BASIC munkaterület törlődik. Elveszik a program és valamennyi változó. Új program írása előtt célszerű a parancs kiadása. Legyünk azonban óvatosak, nincs mód a NEW hatástalanítására.

A C-16 kalkulátorként való használatán kívül a másik legegyszerűbb eset, amikor mások által megírt programot akarunk használni. Ehhez természetesen a rendelkezésünkre kell állnia valamilyen háttértárnak, például kazettás egységnek. A <programnév> nevű programot a

LOAD "<programnév>" (kazettás egység)
DLOAD "<programnév>" (lemezegység)

parancsok segítségével tölthetjük be kazettáról, illetve lemezről. A programot ezután a RUN paranccsal lehet elindítani.

A terminátor billentyűknél már említettük, hogy a <RUN> billentyű lenyomása a lemezegységről betölti és elindítja a katalógusban szereplő első programot. Ilyenkor a RUN parancsot magát már nem kell beírnunk!.

Programok tárolása

A megírt BASIC programokat a memóriából szalagra vagy lemezre másolhatjuk, 'elmenthetjük'. Az így elmentett programokat azután bármikor visszatölthetjük a memóriába. A mágneses adathordozón tárolt program nem vész el. (Ha csak tévedésből felül nem írjuk.) A programot a

```
SAVE "<programnév>"   (kazettás egység)
DSAVE "<programnév>"  (lemezegység)
```

parancsok segítségével másolhatjuk át a kazettára, illetve a hajlékonylemezre (floppy-ra).

A kazettás egység és a lemezegység használatáról a 4., illetve az 5. fejezetben részletesen szólunk.

3. fejezet

C-16 BASIC

3.1 BASIC: összefoglalás

3.1.1 A C-16 BASIC felépítése

A C-16 személyi számítógép a BASIC 3.5-ös jelű változatát tartalmazza. Ez a változat felülről kompatibilis a BASIC V2.0-ás változattal, amelyikkel a Commodore 64 dolgozik. Az új változat lényeges bővitéseket tartalmaz az eredetihez képest. A leglényegesebbek a grafikus utasítások, amelyek a C-64-en egyáltalán nincsenek. Két utasítás teszi lehetővé a hanggenerátor használatát. A lemezegység kezelését is további BASIC utasítások segítik. Végeredményben a C-16 BASIC-je a BASIC V2.0-ás változathoz képest 49 új utasítást tartalmaz.

A C-16 személyi számítógép programozásához az alábbi - a gépbe beégetett - programkomponensek állnak rendelkezésre:

a/ képernyő szerkesztő;
b/ BASIC interpreter;
c/ a perifériák kezelését végző monitor;
d/ gépi kódú monitor.

A képernyő szerkesztőről részletesen a 2.1 paragrafusban szóltunk. Segítségével a képernyő bármelyik sorát átadhatjuk feldolgozásra az interpreternek. Az interpreter azt a sort dolgozza fel, ahol a kurzor állt, amikor a <RETURN>-t megnyomtuk.

A d/ alatt említett gépi kódú monitorról részletesen szólunk a 8. fejezetben.

A BASIC interpreter két részből áll: egyrészt a programok szerkesztését biztosító **programszerkesztő**ből, másrészt a BASIC **futtató rendszer**ből. Ebben a paragrafusban a BASIC programozási nyelvet ismerők számára foglaljuk össze a C-16 BASIC interpreter legfontosabb jellemzőit.

A programokat a

RUN <sorszám>
GOTO <sorszám>
GOSUB <sorszám>

parancsokkal, vagy a **<RUN>** billentyű lenyomásával indíthatjuk el. A GOTO, illetve GOSUB parancsok nem törlik a változók értékét, a többiek igen. Lehetőség van a programokat nyomkövetési üzemmódban elindítani. Ebben az esetben a végrehajtott utasítások sorszámát az interpreter folyamatosan kiírja a képernyőre. A nyomkövetést a **TRON** utasítással kapcsolhatjuk be, illetve a **TROFF** utasítással kapcsolhatjuk ki. A fenti két utasítást a programban is bárhol használhatjuk.

Lehetőség van a programba hibakezelő rutin beépítésére. A hibakezelő rutin belépési pontját a

TRAP [<sorszám>]

utasítás segítségével adjuk meg. A hibakezelő rutinból a **RESUME** utasítással térhetünk vissza. A hibát okozó sor számát az **EL**, a hiba kódját az **ER** fenntartott változók tartalmazzák. A hiba szövegét az **ERR$(N)** tömb elemeinek a segítségével írathatjuk ki. Ha nem használunk hibakezelő rutint, és a program hiba miatt megszakad, akkor a **HELP** parancs kiadása után az interpreter kiírja a hibát okozó sort. A sornak a feldolgozatlan része **villog**.

A futó program a **STOP** vagy az **END** parancs végrehajtásakor áll meg, a program változói **nem törlődnek**. Mindkét esetben a program a **CONT** parancs kiadásával folytatható.

A programszerkesztő hagyományos. A BASIC sorok tetszőleges sorrendben beírhatók, a törlést egy üres sor beírásával érhetjük el. Az **AUTO** parancs automatikus sorszámozást ad. Egy meglévő sor újraírása a szóban forgó sor módosítását eredményezi. A program szövegét a **LIST** paranccsal listázhatjuk ki. A teljes programot a **NEW** paranccsal törölhetjük. Egynél több programsort a **DELETE** parancs segítségével törölhetünk. A **RENUMBER** paranccsal a programot újraszámozhatjuk. A parancs a megfelelő GOTO, GOSUB stb. hivatkozásokat is megfelelően átszámozza.

A **KEY** utasítás segítségével a funkcióbillentyűkhöz szövegeket rendelhetünk. A funkcióbillentyű lenyomása a hozzárendelt szövegrész beírásával ekvivalens.

A kész programokat a **SAVE**, **DSAVE** utasítás segítségével menthetjük kazettára, illetve lemezre. A tárolt programot a **LOAD**, illetve **DLOAD** utasítás segítségével tölthetjük be a memóriába.

Magának a C-16 BASIC-nek két érdekessége van. Egyik, hogy a perifériákra vonatkozó utasítások valamennyi perifériára ugyanazok. (Nincs például külön LPRINT utasítás. A nyomtatóra is a PRINT utasítás segítségével írhatunk.) Másik, hogy nem tesz szigorú különbséget parancs és utasítás között.

A C-16 BASIC nem teszi kötelezővé az értékadó utasításban a LET használatát. A **változók neve** akármilyen hosszú lehet, de megkülönböztetésükre csak az **elsú két karaktert** használhatjuk. Felismeri az **egész**, a **valós** és a **sztring típus**okat. Az egész, illetve a sztring típusok jele a név után írt **%**, illetve **$** jel. A tömbváltozóknak maximum 255 indexe lehet. A tömböket a **DIM** utasításban kell definiálni. Az egyes indexek értéke 0-tól a DIM utasításban megadott értékig terjedhet.

A szorzás, osztás jele: *, illetve /. A hatványozás jele a felfelé mutató nyíl: ^. Logikai kifejezésekben a <, >, <=, >=, =, <> relációs jelek használhatók.

Az egyetlen feltétel nélküli vezérlésátadó utasítás a **GOTO** , ami után csak egy **sorszám** állhat (kifejezés nem). Feltételes vezérlésátadó utasítások az

IF...THEN...[:ELSE...]
ON...GOTO...
ON...GOSUB...

utasítások.

A ciklusutasítás

FOR...TO...[STEP]...

alakú. A ciklusváltozó csak **valós** lehet! A ciklusok tetszőleges mélységben **egymásba skatulyázhatók**. A lépésköz lehet negatív is.

Lehetőség van

DO...LOOP

alaku ciklusok használatára is. A DO, illetve a LOOP utasításokban UNTIL **<logikai kifejezés>**, illetve WHILE **<logikai kifejezés>** alakú utasításokat használhatunk a kilépés feltételének megadására. A ciklusból, annak lejárta előtt, az **EXIT** utasítás felhasználásával léphetünk ki.

Alprogramok használatát a **GOSUB, RETURN** utasításpár teszi lehetővé. Paraméterek átadására nincs lehetőség. A szubrutinhívások tetszőleges mélységben egymásba skatulyázhatók.

A C-16 **ki-/bemeneti utasítás**ainak használatához ismerni kell az ún. elsődleges kimeneti, illetve bemeneti eszköz fogalmát. Elsődleges bemeneti eszköz mindig a **billentyűzet.** Innen várja az inputot a képernyő szerkesztő és az INPUT utasítás. Az elsődleges beviteli eszközről bevitt adatok a képernyőn mindig megjelennek. Az elsődleges kiviteli eszköz a képernyő. A rendszerüzenetek és a PRINT utasításban adott mennyiségek mindig ide íródnak ki. Az elsődleges kiviteli eszköz tetszőleges megnyitott file-ba átirányítható a **CMD** utasítás segítségével. Programhiba után az elsődleges kimeneti eszköz újra a képernyő lesz.

Egykarakteres beviteli utasítás a **GET** és a **GETKEY.** A GET hatására a billentyűzeten éppen lenyomott billentyű ASCII kódja a GET-et követő változóba kerül, mint egy egyhosszúságú sztring. Ha nincs benyomva egyetlen billentyű sem, akkor a változó értéke az üres sztring lesz. A GETKEY ettől abban különbözik, hogy az interpreter **addig vár,** míg le nem nyomunk legalább egy billentyűt.

Az elsődleges kimeneti eszközre való kiírásra a **PRINT** utasítás szolgál. A kiírandó mennyiségek elválasztásánál a vessző (,) és a pontosvessző (;) hatása a szokásos. Ugyancsak használhatók ezek a PRINT utasítás végén. Ha egy PRINT utasítás nem vesszőre vagy pontosvesszőre végzúdik, az utasítás egy CHR$(13) jelet is kiír, így a következő PRINT már egy új sorba nyomtat. Lehetőség van a PRINT USING utasítás használatával értékek megformázott kiírására is.

Adatbevitelre az **INPUT** utasítást használhatjuk. Az input sor - a puffer mérete miatt - legfeljebb **88 karakter**ből állhat. Nincs INPUTLINE utasítás. Ha INPUT utasítással sztringet olvasunk be, az idézőjelet csak akkor kell kiírnunk, ha a sztring elválasztó jeleket (,;:) tartalmaz. Az INPUT utasítás segítségével idézőjelet tartalmazó sztringet nem tudunk beolvasni.

Adatoknak a programban való elhelyezésére a **DATA** utasítást használhatjuk. A DATA-ban elhelyezett sztringeket csak akkor kell idézőjelbe rakni, ha tartalmaznak elválasztó jeleket (,;:), vagy grafikus karaktereket. A DATA-ban elhelyezett adatokat a **READ** utasítás segítségével olvashatjuk be. A **RESTORE** utasításnak lehet paramétere, ezért az olvasást tetszőleges DATA utasítástól folytathatjuk.

A file-ok használatára egységesen az **OPEN**, **CLOSE**, **INPUT#**, **GET#** és **PRINT#** utasítások szolgálnak. A file-ok használatához bizonyos azonosítókat kell megadnunk. Ezek a **logikai file-szám,** a fizikai **hardver szám** és a **másodlagos cím** (vagy megnyitási mód). Minden egyes perifériának rögzített hardver száma van. A másodlagos cím jelentése már egységről egységre változik. A logikai file számot az OPEN utasításban adjuk meg, s attól

kezdve a programban vagy parancsban ezzel azonosíthatjuk a file-t.

A lemezegység kezelésére speciális utasítások szolgálnak. A **DIRECTORY** a képernyőre listázza a lemez katalógusát. A **BACKUP**, illetve a **COPY** utasítások lehetővé teszik a teljes lemez, illetve egyes file-ok másolását. A **SCRATCH** parancs segítségével file-okat törölhetünk a lemezről. A **HEADER** paranccsal a lemezeket formázhatjuk meg. A **COLLECT** parancs teszi lehetővé a lemezek újraszervezését.

A fenti, a lemezegység használatát megkönnyítő utasítások a Commodore BASIC 4.0-ás változatának utasításai, s mind szintaktikailag, mind szemantikailag megegyeznek azokkal.

A BASIC V3.5 teljesen új elemei a grafikus utasítások. Legfontosabb a **GRAPHIC** utasítás, amely segítségével kiválaszthatjuk a kívánt grafikus kijelzési módot. A **COLOR** segítségével állíthatjuk be a keret, a háttér stb. színeit. A **SCNCLR** utasítás felhasználásával törölhetjük a teljes képernyőt. A **SCALE** a nagyfelbontású képernyő léptéké' állítja be. A SCALE 0 kiadása esetén a képernyő léptéke megegyezik fizikai méreteivel, vagyis 320*200-as. A SCALE 1 parancs kiadása után az interpreter 1024*1024-esnek tekinti a képernyőt, s ennek megfelelően számítja ki az egyes pontok fizikai koordinátáit.

A grafikus kurzor mozgatására a **LOCATE** utasítás szolgál. A **DRAW** parancs segítségével rajzolhatunk pontokat, illetve egyenes szakaszokat. A **BOX** utasítás tele téglalapok rajzolására szolgál. A **CIRCLE** segítségével egy ellipszis adott darabját rajzolhatjuk meg. Speciális függvények tájékoztatnak a grafikus kurzor helyéről, a rajzolás színéről stb. A C-16 egy kiváló minőségű **PAINT** utasítással rendelkezik, amelyik segítségével a képernyő egyes részeit *kifesthetjük, besatírozhatjuk*.

A C-16 video chip-je valójában egyetlen dologban tér el a C-64-étől: nincsenek sprite-jai. Ezen segítendő, lehetőség van a grafikus képernyő egy darabjának sztringként való tárolására, illetve az ennek a sztringnek megfelelő képdarabnak a képernyőn **bárhová történő visszahelyezésére**. Erre szolgálnak az **SSHAPE**, illetve a **GSHAPE** utasítások.

A C-16 hanggenerátorának bekapcsolása egy megfelelő **VOL** utasítás kiadásával történik. Ezt követően a **SOUND** segítségével szólaltathatjuk meg az egyes hangokat.

A gépi kódú monitorba a **MONITOR** BASIC parancs kiadásával léphetünk be. A monitor parancsai egykarakteresek, a paramétereket hexadecimális alakban kell megadnunk. A monitor lehetőségeiről a 8. fejezetben szólunk részletesen.

3.1.2 A BASIC interpreter működése

A C-16 képernyő szerkesztője a <RETURN> billentyű benyomását minden esetben úgy értelmezi, hogy befejeztük a BASIC parancs vagy programsor bevitelét, és átadja a vezérlést a BASIC interpreternek. Az interpreter első lépésként meghív egy rutint, amelyik a billentyűzetről bevitt utolsó sort (amely jelenleg a képernyő-memóriában található meg) áttölti az input pufferbe. Erre azért van szükség, hogy a ?"<CLEAR>EREDMENYEK=" típusú utasítások végrehajthatók legyenek. Ha az interpreter az utasításokat a képernyő-memóriából venné ki, a fenti utasítás a <CLEAR> végrehajtása után elveszne. (Ez a probléma természetesen nem merül fel abban az esetben, ha a képernyő egy meghatározott része - például az utolsó sor - szolgál a parancsok, programsorok bevitelére. Ilyen a ZX-81 rendszere.) Az input puffer a 2. lapon helyezkedik el a $0200-$0258 címeken. A fenti rutin az inputsor végére egy 0 byte-ot is elhelyez. Miután az interpreter a bevitt sort elhelyezte az input pufferbe, megvizsgálja, hogy számmal kezdődik-e, vagy sem.

Ha igen, akkor az inputsort programsornak értelmezi, és elhelyezi a memóriában. Ha nem, akkor parancsnak tekinti, és végrehajtja.

Nézzük először azt az esetet, amikor parancsot vittünk be. Az interpreter ennek felismerése után meghívja a **tokenizáló rutint**. Ennek a rutinnak a feladata, hogy az input pufferben található sorban megkeresse a BASIC alapszavakat, ezeket token-jükkel helyettesítse. Ez a rutin ismeri fel a kulcsszavak rövidítését, a ? utasítást és a <pi>-t. Ügyel arra, hogy páros számú idézőjel után szabad csak az alapszavakat tokenjükre cserélni. Az inputsor kódolása általában a sor rövidülését eredményezi.

Ezután az interpreter belép abba az ellenőrző ciklusba, amelyik a tokenizált BASIC szövegeket végrehajtja. Ez a belépési pont nem egyezik meg azzal, ahová a RUN utasítás után lép be az interpreter.

Ha az inputsor programsornak bizonyult, először a sorszám 2-byte-os egész számmá való konvertálására kerül sor. Ha ez hibát eredményez (a szám nagyobb 64000-nél) az interpreter hibajelzést küld, és visszatér a szerkesztőbe. Ha nem, sor kerül a maradék sor tokenizálására az előbb már említett rutin segítségével. Az interpreter ezután ellenőrzi, szerepel-e ilyen sorszámú sor a programban. Ha igen, akkor ezt törli a memóriából és az utána szereplő sorokat lejjeb mozgatja. Ezután meg-

vizsgálja, van-e elég hely az új sor elhelyezésére. Ha nincs, a program végrehajtása ?OUT OF MEMORY ERROR hibaüzenettel megszakad, és a vezérlés visszaadódik a képernyő szerkesztőnek. Ha van hely és az inputsor nem üres, a program a megfelelő helyen annyival tolódik feljebb a memóriában, hogy a sor éppen beférjen. Ilyen módon természetesen a programsorok elején álló - és a következő sor elejére mutató - számok elromlanak. Ezért kerül sor egy további rutin végrehajtására, amelyik újraszámítja ezeket a mutatókat.

Ha az inputsor üres (tehát egyedül a sor végét jelző 0 byte-ból áll) az utolsó lépésekre nem kerül sor. Egy üres sor pl. 213 <RETURN> bevitele így a sor törlését eredményezi.

Utoljára hagytuk a BASIC-kulcsszavak végrehajtását ellenőrző ciklus ismertetését. Két **belépési pont**ja van. Az egyik, amikor parancsot hajtunk végre. A másik belépési pont, amelyen keresztül a RUN parancs végrehajtása során lép be az interpreter. A rutin ellenőrzi, nincs-e lenyomva a <STOP> billentyű, majd sor kerül a következő BASIC-byte olvasására. Megvizsgálja, hogy ez nem sor végét jelző nulla-e. Parancs módban ez a ciklusból való kilépést, a képernyő szerkesztőbe való visszatérést eredményezi. Program végrehajtása közben a nulla a következő sor elején álló mutató ellenőrzését jelenti. Ha ez 0, akkor END nélkül ért véget a program, és a vezérlés visszatér a szerkesztőbe. Ha nem, akkor a program végrehajtása egy új sorral folytatódik. Ha bekapcsoltuk a nyomkövetést, akkor az éppen végrehajtás alatt álló programsor száma kiíródik a képernyőre, ha nem, a végrehajtás azonnal megkezdődik.

Ha nem nulla byte-ot talált az interpreter, akkor megvizsgálja a következő karaktert, amely

a/ kettőspont;
b/ ASCII karakter;
c/ token;
d/ egyéb

lehet. Minden egyes esetben a rutin egy belépési pont címét helyezi a verembe. A belépési pont annak a tevékenységnek az elejére mutat, amit az adott esetben végre kell hajtani. Az a/ esetben nincs tennivaló. A b/ esetben az interpreter LET utasítást tételez fel; s az ennek megfelelő belépési pontot tölti a verembe. A c/ esetben a BASIC elején levő táblázatból megkeresi az ahhoz a tokenhez tartozó belépési pontot, és ezt tölti be a verembe. A d/ esetben a ?SYNTAX ERROR üzenetet kinyomtató hibarutin kezdőcíme töltődik a verembe.

A fenti rutin végén az interpreter egy újabb (gépi kódú) RTS utasítást hajt végre, ami a verembe töltött belépési pontra való ugrást eredményezi.

A belépési pontoknak megfelelő rutinok végén - hacsak nem történt hiba - a vezérlés az ellenőrző ciklus legelejére adódik vissza, függetlenül attól, hogy direkt vagy program módban vagyunk-e.

Külön szólunk a RUN végrehajtásáról. A RUN utasítás végrehajtása egy mutatónak a BASIC szöveg megfelelő helyére való állításával kezdődik. Ezután kerül csak a vezérlés az elenőrzési ciklusba.

A BASIC interpreter működésének ismerete elengedhetetlen a BASIC lehetőségeinek kiterjesztéséhez. A fent jelzett rutinok ugyanis általában egy JMP ($xxxx) gépi kódú utasítással kezdúdnek, ahol $xxxx egy-egy RAM címet jelent (általában a 3. lapon). A címek - hacsak nem írtuk át őket - közvetlenül az indirekt ugrás alá mutatnak vissza, s így folytatják a program végrehajtását. Ugyanakkor a programozónak lehetősége nyílik ezeknek a legfontosabb eljárásoknak az átírására.

Végül pár szót szólunk a C-16 számítógép megszakító rendszeréről, amelyik közvetlenül támogatja a BASIC interpretert.

A számítógép bekapcsolása után, amikor a tápfeszültség eléri a 4.75 V-ot, a számítógép automatikusan generál egy *restart* jelet. Ennek hatására a programszámláló a ($FFFC) mutató értékével töltődik fel, előtte azonban a maszkolható megszakításokat a gép letiltja. A JMP ($FFFC) végrehajtása az I/O regiszterek feltöltésével, majd a legfontosabb rendszerváltozók értékének beállításával kezdődik. Ezt követően kerül sor a memória tesztelésére, illetve a BASIC munkaváltozók beállítására. Legvégül a gép bejelentkezik, kiírja a szabad memóriaterület nagyságát, és a 'READY.' üzenet kiírásával belép a képernyő szerkesztőbe. Ez az inicializálási eljárás állítja be a BASIC interpreter megszakítási rendszerét is.

A hardver megszakító rutin, amely másodpercenként mintegy ötvenszer hajtódik végre, a következő szolgáltatásokat végzi:

a/ a kurzor villogtatása;
b/ a stop billentyű lenyomásának ellenőrzése;
c/ a billentyűzet olvasása;
d/ a 0. lapon található óra aktualizálása;
e/ a megszakítások kezelése.

3.2 A BASIC programozási nyelv

3.2.1 A szintaxis

A BASIC nyelvet gyakran szokás "angol" nyelvűnek is nevezni. Ez a kifejezés teljesen félrevezető, mert igaz ugyan, hogy a BASIC kulcsszavak angol szavak is egyben, de a programok 'nyelvtani szerkezetének' nem sok köze van az angol nyelvhez. A BASIC parancsok, sorok írását éppen úgy meg kell tanulnunk, mint bármely más program nyelvét. De még ebben az esetben is lehetnek értelmezési problémák. Helyesnek tekinthetünk-e egy 10 GOSUB 132 utasítást, ha a 132-es sor nem létezik?! Hogyan fogalmazzuk meg a DATA...RESTORE...READ utasítások egymáshoz való viszonyát? A problémát legegyszerűbben úgy hidalhatjuk át, ha az egyes BASIC utasításoknak külön-külön definiáljuk a szerkezetét. Ebben az esetben a parancs végrehajtása közben legalábbis ?SYNTAX ERROR hibajelzést nem kapunk.

A C-16 BASIC sokban hasonlít a Commodore cég többi gépén futó BASIC-ekhez. Ezek valójában kompatibilisek, nemcsak a forrásnyelv, hanem a gépi megvalósitás szintjén is. Ha valaki elsajátítja a C-16 programozását, az azonnal képes a VC-20, a C-64, a C-128, a PET 600/700-as típusú gépek használatára. Mindegyik BASIC valójában egy egyszerűsített **Microsoft BASIC**. Az egyszerűsítés elsősorban a perifériák kezelésére vonatkozik.

A C-16 BASIC a programsorok, parancsok szövegében levő szóközöket speciálisan kezeli. A szóközök száma a legtöbb esetben érdektelen. Akad azonban néhány kivétel. A file műveletek utasításaiban a kulcsszó és a # jel között **sohasem** lehet szóköz! A programsor elején levő és a sorszámot közvetlenül követő szóközök elvesznek.

Lehetőség van a BASIC kulcsszavak rövidítésére. Ez azt jelenti, hogy csak az alapszó első néhány karakterét gépeljük be, de ezek közül az **utolsót siftelve**. A 3.3 paragrafusban felsoroljuk, hogy melyik alapszót hogyan lehet rövidíteni.

3.2.2 Típusok

A Microsoft BASIC interpreterek három **változótípust** különböztetnek meg; ezek az **egész**, a **valós** és a **sztring** típusok. ***A sztringeket karakterláncoknak vagy karakterfűzéreknek is szokás hívni.*** A sztringek használata teszi lehetővé a szövegek feldolgozását. A különböző típusokkal más és más műveleteket

végezhetünk. A C-16 BASIC ehhez a három típushoz némileg hasonlóan kezeli a **függvénydefiníciókat** is.

3.2.3 Konstansok

A C-16 BASIC háromfajta konstans használatát engedi meg. Ezek azonosak a három típussal:

1/ egész;
2/ valós;
3/ sztring

konstansok.

1/ Az ***egész konstans***ok előjeles egész számok. A pozitív (+) előjel kiírása nem kötelező. A számnak a [-32768, 32767] zárt intervallumba kell esnie.

Példák:

123
-2567
9899

2/ A ***valós konstans***ok a következő részekből állhatnak:

a/ a szám egész része
előjeles egész szám (I);
b/ tizedespont (.);
c/ a szám törtrésze
előjel nélküli egész szám (F);
d/ az exponens jele (E);
e/ az exponens(k)
előjeles egész szám.

A fentiek közül a tizedespont (.), vagy az exponens jelének (E) használata kötelező. Ha ezek hiányoznak, a gép a fenti konstanst egésznek tekinti. Az a/-e/ részekből álló valós konstans értéke, leszámítva a kerekítési hibákat a következő:

$$\left(I + \frac{F}{10^{n}} \right) * 10^{k},$$

ahol n az F törtrész jegyeinek száma. Az a/-c/ részek alkotják a **mantisszát**, a d/-e/ részek pedig az **exponenst**.

A legnagyobb, illetve legkisebb pozitív valós szám, amit az interpreter még tárolni tud:

MAX=1.70141183E+38
MIN=2.93873588E-39

Ha valamilyen számítás eredményeként MAX-nál nagyobb számot kapunk, a végrehajtás ?OVERFLOW ERROR hibaüzenettel megszakad. Ha a számítás eredménye MIN-nél kisebb, a végrehajtás folytatódik: az interpreter a 0.0 értékkel számol tovább.

A valós konstansok alakjának ismerete igen lényeges a nyomtatási kép tervezéséhez. A PRINT utasítás a valós típusú változók értékeit 9 jegyre kerekítve nyomtatja ki. Amennyiben a szám nem kisebb .01-nél és nem nagyobb 999999999-nél, a kiírás nem tartalmaz exponenst. Ha a fenti feltétel nem teljesül, a nyomtatási kép mantisszája mindig 1.-nél kisebb, de 0.1-nél nagyobb szám lesz, s a megfelelő exponens is kiíródik.

Példák:

123.456
2.43
-.99937
+3.1926
235.2E6
235E16
-.2E-13
.1E-2
.0123

3/ A ***sztring*** (vagy ***szöveg***) ***konstans***ok alfanumerikus és grafikus jelek sorozatát jelentik. A sztring konstansok hosszát csak az köti meg, hogy a képernyőről legfeljebb 88 karakter hosszú sort lehet bevinni. A sztring konstansokat idézőjelek (") közé kell írni. Igy az egyetlen alfanumerikus jel, ami nem szerepelhet sztring konstansban, az maga az idézőjel. (Sztring változó értékeként előálló sztringben már szerepelhet, a CHR$(34) felhasználásával.)

Példák:

```
"gyok="
"EREDMENYEK:"
"HOGY VAGY?"
"25.000.000$ ELEG?"
" "
"" (üres sztring, nem egyenlő CHR$(0)-val!)
```

3.2.4 **Változók**

A C-16 BASIC interpreter a változók három típusát különbözteti meg, attól függően, hogy egész, valós vagy sztring mennyiségek tárolására szolgál. Ezen kívül megkülönböztet **egyszerű** vagy **indexes** (**tömb**) változókat. Az interpreter a változók típusát a nevüket követő **$**, illetve **%** jelből állapitja meg. Ezek jelentik a sztring illetve egész változókat. Ilyen jel hiányában az interpreter a változót valós típusúnak tekinti.

A valós és egész típusok közti konverziót a gép automatikusan végzi. Például az L%=L/256 értékadásban L/256-ot egészre kerekíti, ellenőrzi, hogy a [- 32768, 32767] intervallumba esik-e, s ha igen, L%-hez hozzárendeli ezt az értéket. Sztringek és számok közti konverzióra külön beépített függvényeket használhatunk: L$=STR$(L), L=VAL(L$), L%=Val(L$). Két további konvertáló függvény a CHR$ és az ASC. (Lásd ezeket a 3.3 fejezetben!)

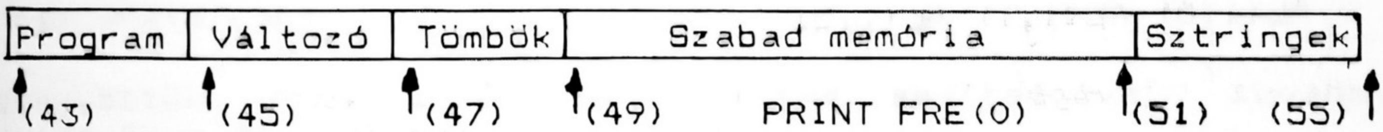
A változó nevek képzésének szabályai a következők:

(i) A változó nevének első karaktere csak betű lehet (A-Z).
(ii) A következő karakter tetszőleges alfanumerikus karakter (0-9,A-Z) lehet.
(iii) Ezt tetszőleges számú alfanumerikus karakter követheti, de ezek nem képezik a név részét. (***KA***TA és ***KA***LAP tehát ugyanaz a név.)
(iv) Ezt esetleg egy $ vagy % követheti; jelezve, hogy a változó sztring, illetve egész típusú.
(v) A következő karakter egy (jel lehet, jelezve, hogy tömbváltozóról van szó. Ezt követik az indexek vesszővel elválasztva és a) jel, amelyek már nem részei a névnek.
(vi) A név nem tartalmazhat egyetlen BASIC alapszót sem (Például ***VA***LI, ***WOR***D nem megengedett nevek). A védett változók TI, ST, TI$, DS, DS$, ER, EL, ERR$ lehetnek változó nevek részei, mert nem számítanak alapszónak (FAN***TI*** megengedett, és egyenlő FA-val).

A változók, akár parancs módban, akár program futása közben használjuk őket, a memóriában tárolt program után helyezkednek el a memóriában. Kivételt a sztringek képeznek. A fenti módon csak a nevük és egy mutató tárolódik. Ez utóbbi a sztring első karakterére mutat. Ezért a sztring-müveletek végrehajtásakor az interpreternek mindig külön meg kell győződnie arról, hogy van-e elég hely a memóriában. Ha nincs, a program futása ?OUT OF MEMORY ERROR hibaüzenettel félbeszakad.

3.2.5 A BASIC program és a változók tárolása

A BASIC program, illetve a változók a memóriában a következőképpen helyezkednek el:



A tárolt program egyes komponenseinek első byte-jaira mutatnak a 0. lapon levő mutatók:

BASIC program kezdete	(43)	($2B)
BASIC terület vége	(55)	($37)
BASIC változók kezdete	(45)	($2D)
BASIC tömbök kezdete	(47)	($2F)
BASIC tömbök vége + 1	(49)	($31)
BASIC sztringek eleje	(51)	($33)

A BASIC munkaterületet elejét, végét a (43), (55) mutatókkal magunk is állíthatjuk. A többi mutató értéke a program szerkesztésétől, illetve futásától függően alakul.

Egyszerű változók tárolása

Tetszőleges, nem tömb változó 7 byte-nyi helyet foglal el a memóriában. Ezen túlmenően a sztring változók - a hosszuknak megfelelően - további byte-okat foglalnak el a BASIC munkaterület végén. Kivételt képeznek a programban szereplő szövegkonstansok. Ezek esetében a sztring változó mutatója a program megfelelő helyére mutat. A tömbök az egyszerű változók után helyezkednek el a memóriában, így például az XT változó első használata azt eredményezi, hogy a tömböket 7 byte-tal feljebb tolja az interpreter, majd az XT változót a helyére rakja. A tárolás nem a legtömörebb, egész és sztring változók esetében 3, illetve 2 felesleges byte is van.

Tömbváltozók tárolása

A tömbváltozók tárolása a tömbre vonatkozó legfontosabb adatok tárolásával (név, dimenziószám, az egyes dimenziók nagysága) kezdődik, majd ezt követik az egyes tömbelemek 5, 3, illetve 2 byte-on, attól függően, hogy valós, sztring vagy egész típusu tömbről van-e szó. A tömbváltozó tartalmaz még egy relativ mutatót is, amelyik a következő tömbváltozó első byte-jára mutat. A tömbelemek oszlopfolytonosan helyezkednek el a memóriában, a DIM A(1,2) deklaráció hatására például ilyen sorrendben:

AL(0,0),AL(1,0),AL(0,1),AL(1,1),AL(0,2),AL(1,2).

Ugyanezeket az elemeket mátrix alakban is elképzelhetjük:

AL(0,0) AL(0,1) AL(0,2)

AL(1,0) AL(1,1) AL(1,2)

3.2.6 Kifejezések

Az algebrában tanultakhoz hasonlóan konstansokból, változókból és egy- illetve többváltozós műveletekből épithetünk fel **kifejezéseket**. A kifejezéseknek két fajtája van:

1/ **aritmetikai**;
2/ **sztring** kifejezések.

Sztring kifejezések

Először a sztring kifejezésekről szólunk, mert ez az egyszerűbb eset. Sztring kifejezések a következők lehetnek:

a/ sztring változók;
b/ két sztring kifejezés a + jellel összekapcsolva;
c/ minden olyan függvény, amelynek a nevében a $ jel szerepel. Ezek: CHR$, ERR$, HEX$, LEFT$, MID$, RIGHT$, STR$. (Természetesen a függvények argumentumainak megfelelő típusú kifejezéseknek kell lenniük.)
d/ a sztring kifejezések tetszés szerint zárójelezhetők.

Példák:

```
"AB CD EF"
G$+"N"
CHR$(N)
STR$(25)+RIGHT$(K$,3,6)
"NEW"+A$+B$.
```

A sztring függvények hatásáról a 3.3 paragrafusban részletesen lesz szó. A + sztring-művelet a sztringek **egymáshoz fűzését, kompozícióját** jelenti. Például:

```
"KUTYA"+"FUL"+"E"="KUTYAFULE";
"ABRAKA"+DABRA"="ABRAKADABRA".
```

Az "" (üres) illetve CHR$(0) sztringek hatása a + **művelet esetén** nem ugyanaz:

```
A$+""=A$
A$+CHR$(0)<>A$
```

Megjegyezzük, hogy a DEF FN utasítás segítségével további sztring függvényeket **nem** definiálhatunk.

Aritmetikai kifejezések

Az aritmetikai kifejezések esete nem csak azért bonyolultabb, mert több művelet végezhető velük, hanem mert az aritmetikai kifejezések - mint speciális esetet - magukban foglalják a logikai kifejezéseket is. A logikai értékeket az interpreter 2-byte-os egész számként tárolja, s a megfelelő logikai műveletet ('és', 'nem', 'vagy' stb.) valamennyi biten egyszerre elvégzi. Ilyen módon logikai műveleteket számok közt is végezhetünk (például X% AND B% értelmes). Az interpreter a (2-byte-os egész számként tárolt) 0-t **hamis**nak, az ettől eltérő értéket **igaz**nak tekinti. **Az összehasonlítások eredményeként** azonban **mindig 0-t vagy -1-et kapunk.**

Relációs jelek

A relációs jelek (<,<=,>=,>,<>,=) segítségével két számot, vagy két sztringet hasonlíthatunk össze. Ha a számok típusa eltérő, az interpreter az egész számot lebegőpontosra alakítja át, és utána hajtja csak végre az összehasonlítást. Ha a két mennyiség eleget tesz a relációnak, eredményül -1-et, ha nem 0-t kapunk.

A relációs jelek értelme, amikor számokat hasonlítunk össze, a szokásos:

```
=    egyenlő
<>   nem egyenlő
>    nagyobb
<    kisebb
>=   nagyobb vagy egyenlő
<=   kisebb vagy egyenlő
```

Sztringek esetén = és <> jelentése értelemszerű. Az **A$<B$** jelentése a következő. Sorba vesszük az A$, B$ sztringekben szereplő jeleket, és megnézzük, melyik az a hely, ahol először eltérnek. Ha ilyen nincs, akkor A$ < B$ pontosan akkor teljesül, ha A$ rövidebb, mint B$. Ha ilyen hely van, akkor A$<B$ azt jelenti, hogy az első eltérő helyen szereplő jel ASCII kódja az A$ sztringben kisebb, mint B$-ban. Elképzelhető, hogy az A$ sztring a B$ sztring kezdőszelete (pl. A$="hazmester", B$="haz"). Ebben az esetben A$<B$ mindig teljesül.

Példák:

```
1=4                hamis (0)
14>=62             hamis (0)
14<62              igaz  (-1)
5*5 <> 16          igaz  (-1)
"ABC"<"AX"         igaz  (-1)
"A"<"D"            igaz  (-1)
"A"<"9"            hamis (0)
"XYZ">"XY"         igaz  (-1)
"XYZU"<="XY9"      hamis (0)
""<=CHR$(0)      igaz  (-1)
CHR$(0)<="@"     igaz  (-1)
```

Az eredményről magunk is meggyőződhetünk. Adjuk ki például a PRINT 1=4 parancsot! Válaszul az interpreter az 1=4 állítás logikai értékét írja ki, ami hamis. Az eredmény tehát 0 lesz.

Logikai műveletek

A BASIC interpreter összesen három logikai műveletet ismer, egy egyváltozós (**NOT**) és két kétváltozós (**AND**, **OR**) műveletet. Hatásukról részletesen a 3.3. paragrafusban szólunk.

Aritmetikai műveletek

A BASIC interpreter a négy alapműveletet, valamint a hatványozás műveletét ismeri. Ezek jele a szokásos:

- \+ összeadás
- \- kivonás, ellentett képzés
- * szorzás
- / osztás
- ^ hatványozás

A gyökvonás az X^(1/Y) kifejezés segítségével végezhető el. Az interpreter az algebrából ismert műveleti hierarchia szerint dolgozik. **Egyenrangú műveletek esetén szigorúan balról jobbra haladva végzi el a műveleteket.** Például A/B/C algebrai alakja:

```
  a
 ---
 b*c
```

A kerekítési hibák miatt az algebrából ismert azonosságok nem mindig teljesülnek!

Aritmetikai függvények

A BASIC interpreter a legfontosabb matematikai függvényeket beépített rutinként ki tudja számítani. Ezek: **ABS, ASC, ATN, COS, DEC, EXP, FRE, INSTR, INT, JOY, LEN, LOG, PEEK, POS, RCLR, RDOT, RGR, RLUM, RND, SGN, SIN, SQR, TAN, VAL.** (A felsorolt 24 függvény közt néhány egészen speciális függvény is van. Ezek hatásáról a 3.3 paragrafusban szólunk részletesen.)

Megjegyezzük, hogy a DEF FN utasítás segítségével további - egyváltozós - aritmetikai függvényeket tudunk definiálni.

Ennyi előkészités után definiáljuk, hogyan is épülnek fel az aritmetikai kifejezések:

a/ tetszőleges egész vagy valós változó aritmetikai kifejezés;
b/ ha aritmetikai kifejezéseket műveleti jelekkel összekapcsolunk, akkor újból aritmetikai kifejezést kapunk;
c/ egy aritmetikai függvény, utána zárójelben a megfelelő tipusú argumentum ugyancsak aritmetikai kifejezés;
d/ két aritmetikai vagy sztring kifejezés összehasonlítása aritmetikai kifejezés;
e/ egy vagy két aritmetikai kifejezés logikai műveletekkel összekötve újból aritmetikai kifejezés;
f/ aritmetikai kifejezések tetszés szerint zárójelezhetők;
g/ a logikai és aritmetikai kifejezések azonosak.

Példák:

```
A+B+.23
C^(D+E)/2
((X-C)^(D-E)/2)*10)+1
K%=1 AND M<>X
NOT (D=E)
K%=2 OR ( A=B AND M<X )
FN DEEK(X)+FN DEEK(X+2)=0
```

A fenti definíció egy igen érdekes sajátosságára hívjuk fel a figyelmet. Aritmetikai kifejezések szerepeltetése a logikai kifejezések közt megszokott. A C-16 azonban azonosítja a logikai és aritmetikai kifejezéseket. Tekintsük például a következő értékadást:

```
L%=ASC(L$)-48+(ASC(L$)>64)*7
```

L$ egy hexadecimális számot tartalmaz karakteres alakban. A fenti értékadás L%-hoz a karakternek megfelelő számértéket rendeli.

Hasonlóan az IF X THEN GOTO... utasításban a kifejezés akkor lesz igaz, ha X értéke 0-tól különbözik.

Műveletek sorrendje

Mindig a magasabb rangú műveletek hajtódnak először végre. Egyenrangú műveletek esetén a műveletek végzése balról jobbra halad. A műveletek a következő sorrendben kerülnek végrehajtásra:

```
1/ ^     hatványozás
2/ -     ellentett képzés
3/ * /   szorzás, osztás
4/ + -   összeadás, kivonás
5/ >=<   összehasonlítás
6/ NOT   logikai nem
7/ AND   logikai és
8/ OR    logikai vagy
```

Példák (összefoglaló):

	Aritmetikai	Sztring
Konstans	2.3	"ABCD"
	-5	"EREDMENY"
	2E-1	"DRAGA"
	3.1415	
Változó	XY	AC$
	A(23)	TI$
	X(5,I+7)	UZEN$(J,8)
Kifejezés	A+B	" "+G$
	2*X*X-3*Y	LEFT$(" "+C$,4)
	COS(X)-TAN(4)	MID$(A$,2,3)+C$
	FN LG(X)+ D(I)	

3.3 BASIC alapszavak ABC sorrendben

Ez a fejezet tartalmazza a C-16 BASIC interpreter utasításait és függvényeit, azok leírását, működésük részletes ismertetését. Az egyes utasításokról szóló részek a következőképpen épülnek fel.

Rövidítés: A legtöbb BASIC alapszót nem kell teljes egészében beírnunk, elég az első néhány karakter beírása, amelyek közül az utolsót siftelve (emelve) kell beírnunk. A fenti címszó alatt a rövidítés 'kisbetűk/nagybetűk' karakterkészlet használata esetén látható alakját adjuk meg. A 'nagybetűk/grafikus jelek' karakterkészlet esetében a rövidítés utolsó karaktere egy grafikus jel lesz.

Token: A BASIC interpreter az alapszavakat egyetlen byte-on tárolja. Ez a byte a szóban forgó **alapszó tokenje**. Az érvényes tokenek 128 és 253 közé esnek. A fenntartott változóknak nincs külön tokenje.

Funkció: Ebben a részben röviden összefoglaljuk az utasítás hatását. A BASIC-et ismerők számára az már általában elegendő az utasítás használatához. Kezdők feltétlenül tanulmányozzák át a példákat, illetve az azt követő magyarázatokat.

Szintaxis: Az egyes **szintaktikus egység**ek megnevezéseit **csúcsos zárójel**ek közé tesszük, például <aritmetikai kifejezés>. A **szögletes zárójel**ek közti részek **nem kötelező**ek, opcionálisak. Ha viszont szerepelnek, akkor csak a szögletes zárójelben megadott formában használhatjuk őket. Például LIST [-<sorszám>].

A **kapcsos zárójel**ek közt, egymás alatt szereplő részek **kötelező választási** lehetőségeket sorolnak fel. (Lásd például az ON utasításnál.)

A szintaxis leírásánál szereplő **többi jel** (",: stb.) az **utasítás része**. A szintaxis leírásánál használt szögletes **és** kapcsos zárójelek természetesen **nem** részei az utasításoknak.

Példák: A példákat igyekeztünk úgy összeválogatni, hogy az utasítás legtipikusabb használati lehetőségeit bemutassák. Itt emeljük ki azokat a hatásokat is, amelyek nem szokásosak, **eltérnek** a legtöbb BASIC megvalósítástól. A példákat a végrehajtásra utaló megjegyzésekkel zárjuk.

Hibalehetőségek: Itt soroljuk fel a leggyakoribb hibákat.

ABS

Röviditès: aB *Token:*$BA(182)

Mód: mind parancs, mind program módban használható.

Az ABS jelet zárójelben követő aritmetikai kifejezés abszolút értékét számolja ki.

Szintaxis: ABS (<aritmetikai kifejezés>)

Példák

```
(i)    465 IF ABS(X-X1)<EPS THEN PRINT "** VEGE **"
(ii)   IF ABS(QY)>E7 THEN PRINT"**QY TUL NAGY **"
(iii)  148 IF ABS(X%)<3 THEN GOTO 232
```

Az első példa egyenletek közelítő megoldásában igen gyakori. Ellenőrzi, hogy a gyök X és X1 közelítései elég közel vannak-e egymáshoz. Ha igen, a program megáll, különben tovább folytatja a számítást.

A második példa az ABS direkt módban való használatát szemlélteti. A program elinditása előtt ellenőrizzük QY értékét. Ha az meghaladja az 1E7 értéket, a gép egy **QY TUL NAGY** üzenet kiírásával figyelmeztet. Az üzenet természetesen magába a programba is beépíthető.

A harmadik példa egy NIM játékprogramból való. A gép véletlenszerűen generál gyufaszálakból álló kupacokat. Ha tul kevés (<3) gyufa van egy kupacban, újból generál egy számot.

Hibalehetőségek Ha a zárójelben rossz vagy nem aritmetikai típusú kifejezés áll, akkor különféle hibajelzések generálódhatnak, például szintaktikus hiba, nem megfelelő típus jelzése stb. Ha az aritmetikai kifejezés kiértékelése túlcsordulást okoz, egy ?OVERFLOW ERROR hibaüzenetet kapunk.

AND

Rövidítés: aN *Token:* $AF(175)

Mód: mind parancs, mind program módban használható.

Két kifejezés logikai 'és'-ét határozza meg. Az AND művelet egész számokra is értelmezhető. Az 'és' művelet ekkor bitenként, egymástól függetlenül hajtódik végre. Az AND művelet művelettáblája a következő:

AND	igaz	hamis
igaz	igaz	hamis
hamis	hamis	hamis

Szintaxis: <aritmetikai vagy logikai kifejezés> AND <aritmetikai vagy logikai kifejezés>

Példák:

```
(i)   PRINT -237 AND 750 : REM = 514
(ii)  OK=X>Y AND Y^Z>20 AND X>15
(iii) IF 0<X AND X<6 THEN GOTO 200
```

Az első példában azt mutatjuk be, mi történik, ha aritmetikai kifejezéseket kapcsolunk össze AND-del. A két szám 16-bites megfelelője: 0000 0010 1110 1110 illetve
1111 1111 0001 0010
Az AND műveletet bitenként elvégezve igy eredményül 0000 0000 0010 0010 adódik, ami 514. Az első példa igy végül is 514-et nyomtat.

A következő két példa azt mutatja be, hogyan használhatjuk az AND műveletet összetett logikai kifejezések képzésére. A második példában az OK egész változó értéke -1 (igaz) lesz abban az esetben, ha X>Y, Y^Z>20, illetve X>15 egyaránt teljesül. Különben OK értéke 0 (hamis) lesz.

Az utolsó példa az AND használatát egy összetett feltételes utasításban szemlélteti. A vezérlésátadásra csak abban az esetben kerül sor, ha mind a két feltétel teljesül.

Hibalehetőségek Ha az operandusok kiértékelése közben hiba történik, a hiba típusának megfelelő jelzést kapunk. Gyakori hiba forrása a műveletek sorrendjének pontatlan ismerete. A C-16 BASIC műveleti hierarchiája megegyezik a FORTRAN és az ALGOL programnyelvekével. Először a NOT, azután az AND, legvégül az OR hajtódik végre. Ügyelnünk kell arra, hogy az interpreter először

az aritmetikai műveleteket hajtja végre, s csak azután a logikaiakat.

ASC

Rövidítés: aS *Token*: $C6(198)

Mód: mind parancs, mind program módban használható

Az ASC jelet zárójelben követő sztring kifejezés első karakterének C-16 ASCII kódját adja. Az ASC függvény használata azokban az esetekben gyakori, amikor egy karakter kódját (ami egy szám) könnyebb kezelni, mint magát a karaktert.

Szintaxis: ASC(<sztring kifejezés>).

Példák:

```
(i)    PRINT ASC("A") : REM =65
(ii)   100 X=ASC("ZEBRA")
(iii)  10 GETKEY JEL$
       20 JEL=ASC(JEL$): IF JEL=13 THEN RETURN
(iv)   200 JEGY=ASC(KOD$)-48+(ASC(KOD$)>64)*7
```

Az első példa az ASC függvény hatását mutatja. Az A betű helyére bármilyen karaktert (betűt, számot, grafikus jelet) téve megkapjuk annak ASCII kódját.A második példa eredményül 90-et ad, (X=90) lévén Z ASCII kódja 90.

A (iii) alatti programrészlet azt mutatja, hogy az ASC függvény segítségével a billentyűzetről kapott információt (JEL$) hogyan elemezhetjük. Ezzel az eljárással a teljes billentyűzetet ellenőrizhetjük, kivéve a <STOP> billentyűt. Az ASC függvény helyettesíthető a CHR$ függvénnyel. A JEL=13 helyett rögtön JEL$=CHR$(13)-t kérdezhetnénk. Ekkor persze JEL kiszámítása teljesen felesleges. Más a helyzet, ha JEL$ nem egy 1 hosszuságu sztring. Ekkor csak a 20. sorban látható módszert választhatjuk.

A (iv) alatti példánk egy hexadecimális-decimális konvertáló program része. Ha a KOD$ első karaktere hexadecimális számjegy, akkor JEGY értéke ez a számjegy lesz. Ha például KOD$="B89A", akkor JEGY=11 a B "számjegy" értéke.

Hibalehetőségek A kifejezés kiértékelése közben számos hiba fordulhat elő, köztük az, hogy az eredménysztring hossza meghaladja a 255-öt. Ha a kifejezés nem sztring típusú, akkor *nem megfelelő típus* (?TYPE MISMATCH) üzenetet kapunk. Gyakori

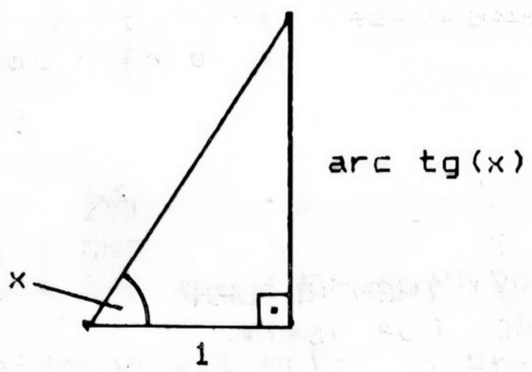
hiba, hogy ASC-t hajtunk végre az üres ("") sztringre. Ekkor ?ILLEGAL QUANTITY hibajelzést kapunk. Ezt elkerülhetjük, ha ASC(J$+CHR$(0))-t használunk ASC(J$) helyett, vagy előre lekérdezzük , hogy az argumentum sztring nulla-e.

ATN

Rövidítés: aT *Token*:$AF(175)

Mód: mind parancs, mind program módban használható.

Az argument arkusz tangensének főértékét számítja ki radiánban. Az alábbi ábra illusztrálja x és arc tg x kapcsolatát:



Szintaxis: ATN(<aritmetikai kifejezés>)

Példák:

```
(i)     PRINT ATN(1): REM =pi/2
(ii)    PRINT ATN(TAN(2*<pi>+<pi>/4)): REM = pi/4
(iii)   ALFA=ATN(AV/BV): BETA=<pi>/2-ALFA
(iv)    FI=ATN(Y/X): R=SQR(X*X+Y*Y)
```

Az első két példában az ATN hatását mutatjuk be. Mivel az ATN függvény az arkusz tangens főértékét számítja ki, ezért a (ii) példában a végeredmény nem egyezik meg a TAN argumentumával.

Az utolsó két példa geometriai jellegű. A (iii) az AV, BV befogójú derékszögű háromszög szögeit számítja ki. A (iv) programsor egy pont (X,Y) derékszögű koordinátáit számítja át polárkoordinátákba. R a pont origótól mért távolsága, FI a pont helyvektorának az X tengely pozitív felével bezárt szöge.

A fenti példákban ALFA, BETA és FI értéke mindig a (-pi/2, pi/2) intervallumba esik. Ha ezt az értéket megszorozzuk 180/pi-vel, az eredményt szögekben is megkaphatjuk.

Hibalehetőségek Az aritmetikai kifejezés kiértékelése közben számos hibalehetőség adódik. Ha a kifejezés kiértékelhető, és nem haladja meg a gépben ábrázolható valós számok felső határát, akkor az ATN rutin már hibajelzés nélkül végrehajtódik.

AUTO

Rövidítés: aU *Token:* $DC(220)

Mód: csak parancs módban használható.

Automatikus sorszámozást biztosít programíráshoz.

Szintaxis: AUTO [<aritmetikai kifejezés>]

Az aritmetikai kifejezés értéke a sorszámozás növekményét adja meg. Az automatikus sorszámozás megszüntetését az AUTO parancs paraméter nélküli kiadásával érhetjük el. Az aritmetikai kifejezés értéke nem kell, hogy egész legyen, ilyenkor a kifejezés egész értéke lesz a sorszámnövekmény.

Példák:

```
(i)   AUTO 5
      10 INPUT A,B
      C=SQR(A*A+B*B)
      PRINT "C=";C
      <RETURN>
      AUTO
```

A példában egy három sorból álló egyszerű programot írtunk meg. Az AUTO 5 parancs kiadásával beállítottuk az automatikus sorszámozást. Ez az első programsor (a 10-es sorszámu) begépelése után azt eredményezi, hogy a következő sor elején a 15-ös sorszám automatikusan megjelenik. A második programsor beírása után a következő sor elejére kiíródik a 20-as sorszám. Ha már nem akarunk több programsort írni, akkor a <RETURN> megnyomásával kilépünk a programírásból, s egy paraméter nélküli AUTO paranccsal kikapcsoljuk az automatikus sorszámozást. Ha ezután írunk be egy programsort, akkor a következő sor elejére már nem íródik ki a sorszám, s a kurzor a sor elejére kerül.

Az automatikus sorszámozást a program első változatának a beírásánál, vagy nagyobb programrészek hozzáírásánál használjuk. Programok javításakor általában nem célszerű az automatikus sorszámozás használata.

Hibalehetőségek Az aritmetikai kifejezés kiértékelése közben számos hibalehetőség adódhat. Ha a kifejezés értéke nem pozitív, vagy túl nagy, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

BACKUP

Rövidítés: bA *Token*:$F6(246)

Mód: mind parancs, mind program módban használható.

Duál lemezegység (CBM4040) használata esetén az egyik meghajtóban levő lemez tartalmát teljes egészében átmásolja a másik meghajtóban levő lemezre. Ily módon adattartalmát tekintve két **teljesen azonos** lemez jön létre. Egymeghajtós lemezegység esetén nem használható!

SZintaxis:

```
                                          { ON }
BACKUP D<arit.kif.1> TO D<arit.kif.2> [  {    } U<arit.kif.3> ]
                                          { ,  }
```

Az első aritmetikai kifejezés a **másolandó** lemezt tartalmazó meghajtó száma, míg a második a létrehozandó új lemezt tartalmazó meghajtó száma. Mindkettő értéke vagy 0, vagy 1 lehet. A harmadik aritmetikai kifejezés értéke a használandó lemezegység hardver egységszáma. Ez 8,9,10 vagy 11 lehet. Ha nem adjuk meg, az interpreter 8-nak veszi. AZ új lemeznek nem kell már megformázottnak lennie, a BACKUP újraformázza a lemezt.

Példák

```
(i)     BACKUP D0 TO D1: PRINT DS$
(ii)    BACKUP D1 TO D0 ON U9
        BACKUP D1 TO D0, U9
```

Az első példa a BACKUP leggyakoribb használatát mutatja: parancs módban a lemezről biztonsági másolatot készítünk, majd lekérdezzük a lemezegység hibacsatornáját. Ugyanezt gyakran szokás programból is használni a következő programrész segitségével:

```
1000 PRINT "KEREM A BACKUP LEMEZT AZ 1-ES MEGHAJTOBA!!!"
1005 GETKEY A$
1010 BACKUP D0 TO D1
1015 PRINT DS$: END
```

A (ii) alatti két parancs az U paraméter két lehetséges írásmódját mutatja.

Hazánkban jelenleg nem forgalmaznak a C-16-hoz (vagy a VC-20-hez, illetve a C-64-hoz) közvetlenül kapcsolható duál meghajtót. A legelterjedtebb lemezegység, a VC 1541-es, egymeghajtós. Ezért teljes lemezek másolásához speciális programokat kell használni.

Hibalehetőségek Egymeghajtós lemezegységek esetén az egység természetesen nem kezdi el a másolást. Ha a másolás közben olvasási vagy íráshiba történik, akkor a másolás félbeszakad. A hibacsatorna olvasásával megtudhatjuk, hogy melyik lemez melyik blokkjával milyen hiba történt. A BACKUP DO TO DO parancs a lemezt önmagára másolja vissza. Ez a mágnesezés felfrissítésére jó, de ilyenkor is célszerűbb új lemezre másolni.

BOX

Rövidítés:bO *Token*:$E1(225)

Mód: mind parancs, mind program módban használható.

A nagyfelbontású képernyőre egy téglalapot rajzol. Az utasítás paraméterei a téglalap nagyságát, dőlésszögét, színét határozzák meg. Megadhatjuk továbbá, hogy a téglalap belsejét is ***befesse-e*** az interpreter vagy sem.

Szintaxis:

BOX [<színtípus>],<x1>,<y1>,<x2>,<y2> [<szög> [,<töltés>]

A <színtípus> a lehetséges négy színtípus (0-3) egyike lehet. Ha elmarad, az 1-nek, az *írás* színének felel meg. <x1>,<y1> a téglalap bal felső, míg <x2>,<y2> a téglalap jobb alsó csúcsának a koordinátái. <szög> a téglalap oldalainak a vízszintessel bezárt szögét adja meg radiánokban. A <töltés> értéke 0 vagy 1 lehet. Ha 0, akkor az utasítás csak a téglalap határát rajzolja meg. Ha 1, akkor a téglalap belsejét is befesti a kiválasztottt <színtípus>-nak megfelelő színnel.

Példák:

```
(i)   GRAPHIC 1,1: BOX 1,10,10,309,189,0,0
      GRAPHIC 1,1: BOX ,10,10,309,189
      GRAPHIC 1,1: LOCATE 309,189: BOX ,10,10
(ii)  GRAPHIC 1,1
```

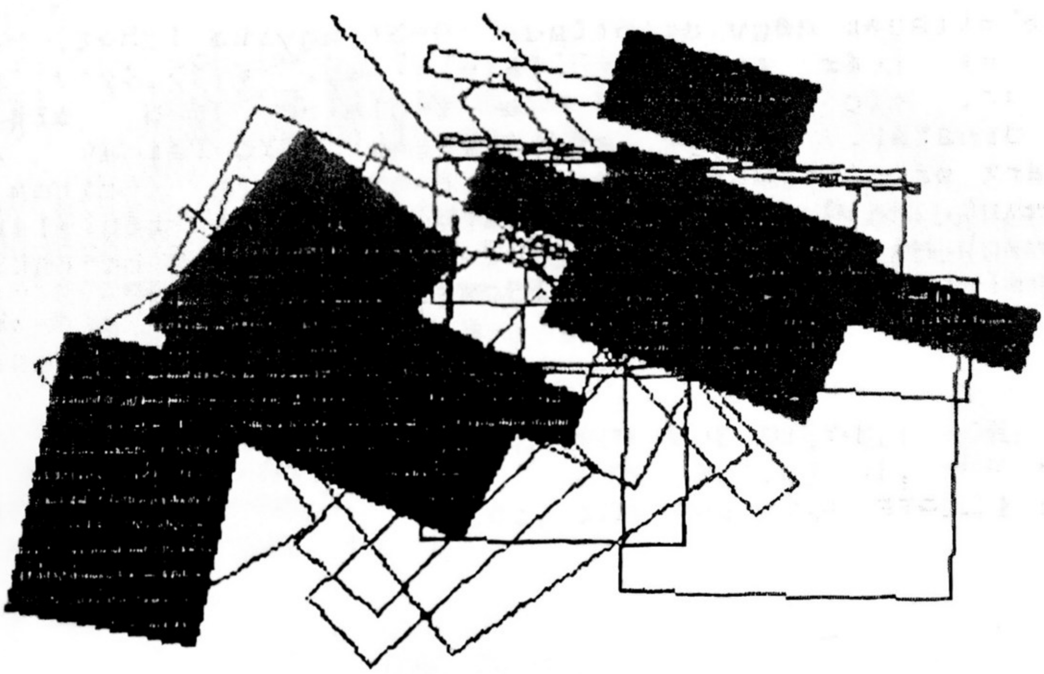
```
      BOX 1,10,10,80,80
      BOX 0,10,10,80,80
(iii) 10 GRAPHIC1,1
      20 FOR I=1 TO 30
      30 A1=RND(0)*160:B1=RND(0)*100
      40 A2=RND(0)*160:B2=RND(0)*100
      50 IF RND(0)>.8 THEN X=1: ELSE X=0
      60 BOX 1,A1+30,B1+10,A1+A2+30,B1+B2+10,60*RND(0),X
      70 NEXT I
```

Az első példa mindhárom sora ugyanazt végzi el: a képernyőre rajzol egy téglalapot. Az első sorban valamennyi paramétert szerepeltetjük. Az utasítás második formájában három paraméter is hiányzik. A harmadik esetben a jobb alsó csúcspont koordinátái is hiányoznak. Ebben az esetben az interpreter a grafikus kurzor pillanatnyi helyzetét használja. Ennek megfelelően a LOCATE utasítás segítségével a grafikus kurzort a leendő téglalap jobb alsó csúcsára állítjuk.

A (ii) alatti példa a színtípus használatát mutatja be. Az első BOX utasítás rajzol egy téglalapot a képernyőre. Ezután ugyanazt a téglalapot rajzoljuk meg, de most a háttér színével. Ez az előző téglalap **törlését** eredményezi.

A (iii) alatti program véletlenszerűen kiválasztott téglalapokat rajzol a képernyőre.

Hibalehetőségek Az aritmetikai kifejezések kiértékelése közben számos hiba történhet. A téglalap nem kell, hogy teljes egészében a képernyőre essen, ebben az esetben az interpreter a téglalap képernyőre eső darabját rajzolja csak meg. A használt színtípusnak összhangban kell lennie a kiválasztott grafikus móddal. Ha nem, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk. A paraméterek elhagyása esetén a megfelelő számú vesszőt ki kell tenni, különben a paraméterek összekeverednek.



CHAR

Rövidítés: chA *Token*: $

Mód: mind parancs, mind program módban használható.

Segítségével mind a karakteres, mind a nagyfelbontású képernyő tetszőleges helyére írhatunk karaktereket.

Szintaxis:
CHAR [<színtípus>],<x>,<y> [,<sztring kif.>] [,<inverz-jel>]]

A <színtípus> a kiírandó karakterek színét adja meg. Ha elhagyjuk, az az 1-es színtípusnak (az írás színe) felel meg. Az <x>,<y> aritmetikai kifejezések a kiírás kezdőpozícióját határozzák meg, amit mind karakteres, mind nagyfelbontású képernyő esetén karakterhelyekben kell megadni. Az <inverz-jel> aritmetikai kifejezés értéke 0, vagy 1 lehet. Ha 1, a sztring kifejezés karakterei inverz alakban kerülnek a képernyőre. Nagyfelbontású képernyő használata esetén a vezérlő karakterek helyett is grafikus jelek íródnak ki.

Példák:

```
(i)    GRAPHIC 0,1: CHAR 1,10,10,"ABRAKADABRA"
       GRAPHIC 1,1: CHAR 1,10,10,"ABRAKADABRA"
(ii)   GRAPHIC 1,1: CHAR 1,10,10,"KUTYAFULE",1
       GRAPHIC 1,1: CHAR 1,10,26,"KUTYAFULE"
(iii)  10 GRAPHIC 1,1
       15 CHAR 1,13,12,"I "+CHR$(211)+" COMMODORE-16"
       20 BOX ,100,92,235,106
       25 GETKEY A$: GRAPHIC 0
```

Az (i) alatti két parancs a karakteres, illetve a nagyfelbontású képernyő használatát mutatja be. Jól látható, hogy a nagyfelbontású képernyő használata esetén is a 10 karaktersort, nem pedig rasztersort jelent.

A (ii) alatti példa a nagyfelbontásu képernyőre írja a "KUTYAFULE" feliratot először inverz, azután normál alakban. A (iii) példa mutatja, hogy sztringkifejezések is használhatók az utasításban.

Hibalehetőségek A színtípus értéke (0-3) csak a kiválasztott grafikus módnak megfelelő lehet. Ha ettől eltér, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk. Ugyanez vonatkozik az <inverz-jel>-re is. (értéke 0, vagy 1 lehet.)

CHR$

Rövidítés: cH (a $ már nem kell!) *Token:*$C7(199)

Mód: mind parancs, mind program módban használható.

Egy tetszőleges, a 0<=x<=255 intervallumba eső számból előállít egy egyhosszúságu sztringet, amelynek ASCII kódja éppen a szóban forgó szám. A C-16 BASIC-ben ez az egyetlen kényelmes eljárás bizonyos speciális karakterek (például <RETURN> és ") kezelésére (CHR$(13), illetve CHR$(34)).

Szintaxis: CHR$ (<aritmetikai kifejezés>)

Az aritmetikai kifejezés nem kell, hogy egész legyen. Ilyenkor a kifejezés egész részével számol tovább az interpreter.

Példák:

```
(i)   A$=CHR$(34)+CHR$(20)+CHR$(30)+CHR$(18)+
           "NEV"+CHR$(146)+CHR$(34): ?A$
(ii)  XE$=CHR$(160)+X$
(iii) PRINT#4,CHR$(7);
(iv)  PRINT#4,CHR$(27);"FOPROGRAM"
```

Példáink azt mutatják be, hogyan lehet a CHR$-t speciális szerkesztő karakterek nyomtatására felhasználni.(Ezeket a karaktereket különben igen nehéz előállítani.)

Az első példa idézőjelek közé helyez egy sztringet, előtte azonban eléje és utána helyezi a <RVSON> illetve a <RVSOFF> karaktereket. A ?A$ hatására idézőjelek közt jelenik meg a NEV negatív (inverz) képe.

A (ii) példa az X$ sztring elejére egy siftelt szóköz karaktert illeszt. Bizonyos perifériák a sztring elején álló szóközöket nem nyomtatják. Ezt XE$=" "+X$ alakban is begépelhetnénk, de akkor nem látszik, hogy az idézőjelek közt siftelt szóköz van.

A harmadik példa elküld a nyomtatónak egy 'bell' jelet. Bizonyos típusú nyomtatók ilyenkor valóban adnak valamilyen hangjelzést.

A negyedik példa ugyancsak a nyomtatóhoz kapcsolódik; a vezérlő karakter hatására a FOPROGRAM karakterei duplaszéles formában nyomtatódnak ki (CBM típusú nyomtatókon legalábbis).

A CHR$ az ASC inverze. Speciális felhasználási területe a különböző karakterkészletek közti konverzió. A PRINT utasítást használó programok nem írhatók át közvetlenül PRINT#4-re. Előbb a kinyomtatandó értékek részletes vizsgálata szükséges, hogy a

nyomtatási kép megegyezzen a képernyőn láthatóval.

CHR$(0) egy **nulla karaktert** jelent, de a hossza 1. A nyomtatási képben ennek azonban nincs nyoma. Legyen például Y$="123"+CHR$(0)+"321". A PRINT Y$ utasítás hatására az 123321 sorozat íródik ki, de LEN(Y$)=7 és VAL(Y$)=123!

Hibalehetőségek Az argumentum kiértékelése, illetve nem megfelelő értéke számos hibajelzést eredményezhet. A speciális karakterek kódjainak rossz használata esetén a kívánt hatás elmarad.

CIRCLE

Rövidítés: cI *Token:* $E2(226)

Mód: mind parancs, mind program módban használható.

Az utasítás segítségével a nagyfelbontásu képernyőre ellipszist, illetve egy ellipszis meghatározott darabját rajzolhatjuk.

Szintaxis:
CIRCLE [<színtipus>],[<x>,<y>],<xr>,[<yr>],[<ks>],[<vs>],[<sz>],[<nv>]

A színtípus a megrajzolandó ellipszis pontjainak a színét határozza meg. Ha elhagyjuk, akkor az interpreter az 1-es színtípust használja. Az <x>,<y> koordinátapár az ellipszis középpontját, <xr>,<yr> a két tengelyét, <sz> pedig az ellipszis nagytengelyének a vízszintessel bezárt szögét határozza meg. Ha az <x>,<y> értékeket elhagyjuk, az interpreter a grafikus kurzor aktuális értékét tekinti az ellipszis középpontjának. Ha <yr> hiányzik, akkor az interpreter az <yr>=<xr> értékkel dolgozik, vagyis kört rajzol. Ha <sz> hiányzik, akkor az interpreter a 0 értékkel dolgozik. Az ellipszis egyik nagytengelye tehát vízszintes, a másik függőleges lesz.

A további paraméterek a rajzolás módját határozzák meg. **Az interpreter az ellipszis egymástól - a középpontjából nézve - <nv> szögtávolságra levő pontjait rajzolja meg, és ezeket egyetlen szakasszal köti össze.** Igy ha <nv>-t 120-nak választjuk, akkor az 'ellipszis' egy háromszög lesz! Ha <nv>-t nem adjuk meg, akkor az interpreter 2-vel számol. A <ks>,<vs> értékek az ellipszis (illetve a neki megfelő töröttvonal) első, illetve utolsó pontjának az ellipszis nagytengelyétől mért szögtávolságát adják meg. Ha például <ks>=0 és <vs>=90, akkor az interpreter csak egy negyed ellipszist rajzol. Ha nem adjuk meg ezeket a paramétereket, akkor az interpreter a <ks>=0, <vs>=360

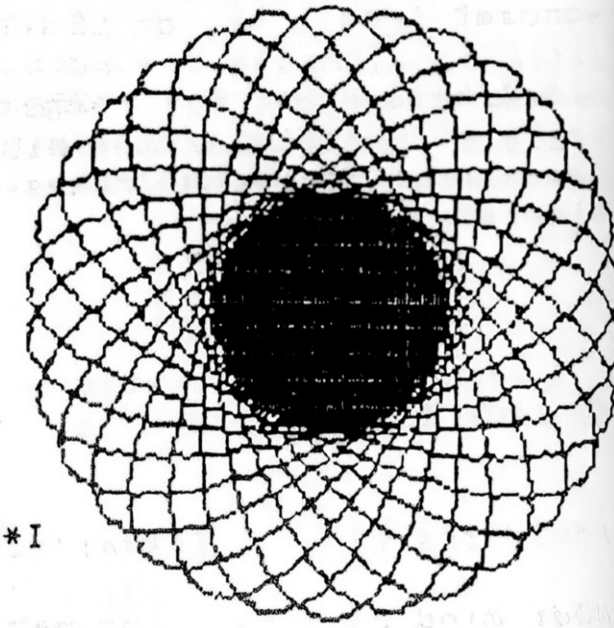
értékekkel számol, azaz a teljes ellipszist megrajzolja. Valamennyi szöget fokokban kell megadni.

Példák:

(i)
```
GRAPHIC 1,1
CIRCLE 1,160,100,45,45
CIRCLE 1,160,100,48,100
CIRCLE 0,160,100,48,100
CIRCLE 1,160,100,48,100,,90
CIRCLE 1,100,100,40,40,,,120
CIRCLE 1,100,100,50,50,,,45
```

(ii)
```
GRAPHIC 1,1
LOCATE 160,100
CIRCLE 1,,45,45
```

(iii)
```
10 GRAPHIC 1,1
20 FOR I=0 TO 14
30 CIRCLE 1,159,100,30,80,,,12*I
40 NEXT I
50 PAINT 1,159,100
60 GETKEY A$
70 GRAPHIC 0
```



Az (i) alatti parancsokat egymás után kiadva a CIRCLE valamennyi lehetőségét kipróbálhatjuk. Az utolsó három parancs azonban nem ellipszist, hanem négyzetet, háromszöget, illetve hatszöget fog rajzolni, az <nv> paraméter megválasztása miatt.

A (ii) alatti példa az (i) alatti első ellipszis rajzolását ismétli meg oly módon, hogy a kör középpontja a grafikus kurzor helye, amit előzőleg egy LOCATE utasítás segítségével beállítottunk.

A (iii) példa egy néhány körből álló rajzot készit.

Hibalehetőségek A <színtípus> értéke (0-3) összhangban kell, hogy legyen a grafikus üzemmóddal. Ha nem, ?ILLEGAL QUALITY ERROR hibajelzést kapunk. Az aritmetikai kifejezések kiértékelése közben számos hibalehetőség adódik. A paraméterek értéke nem lehet negativ.

CLOSE

Rövidítés: clO *Token*:$A0(160)

Mód: mind parancs, mind program módban használható.

Befejezi egy adott file feldolgozását, törli a file-t a három file-leíró táblából. (A háttértárakon levő file-ok tartalma természetesen nem vész el.) A billentyűzetre és a képernyőre megnyitott file-okkal a táblákból való törlésen kivül semmi egyéb nem történik. A többi file esetében a tevékenység attól függ, hogy milyen utasítással nyitottuk meg a file-t.

Szintaxis: CLOSE <aritmetikai kifejezés>

Példák:

```
(i)   OPEN 4,4 : PRINT#4,"HOGY VAGY?": CLOSE4
(ii)  OPEN 1,1,1,"FILE":PRINT#1,"HOGY VAGY?":CLOSE1
(iii) PRINT#4 :CLOSE 4: REM LEZARJA A NYOMTATOT CMD UTAN
(iv)  CLOSE 1,2,3,"$": REM UGYANAZ, MINT CLOSE 1
```

Az (i) példában megnyitjuk a nyomtatót, kiírjuk a "HOGY VAGY?" sztringet, majd a file-t lezárjuk. A (ii) példa ugyanezt a sztringet írja ki a szalagon éppen megnyitott "FILE" nevü file-ba. A (iii) példa mutatja, hogy CMD használata után hogyan kell egy file-t lezárni.

A (iv) példa a C-16 BASIC egy furcsaságára utal, a CLOSE utasítás szintaxisát ugyanaz a rutin ellenőrzi, mint az OPEN-ét. Ezért a CLOSE-ban szerepelhetnek további paraméterek, de azok semmire sem jók. Logikai hibát okozhat viszont a CLOSE 1,2 alak. Ez ugyanis **csak** az 1-es logikai file-t zárja le!

Hibalehetőségek Elsősorban írásra megnyitott lemezes file-ok esetén lényeges, hogy ezeket a file-okat lezárjuk. Ha ezt nem tesszük meg, az utolsó rekord sáv/szektor mutatója nem a megfelelő helyre mutat, s előbb vagy utóbb két file egymásra íródik. Programhiba következtében gyakran maradnak nyitva file-ok. Ilyenkor célszerű a file-okat paranccsal lezárni. Adjuk ki az OPEN 15,8,15,"I":CLOSE 15 parancsot. Amennyiben ez sem zárja le azokat, akkor nincs mit tenni.

CLR

Rövidítés: cL *Token*:$9C(150)

Mód: mind parancs, mind program módban használható.

Valamennyi egyszerű és tömb változót törli a memóriából, magát a BASIC programot azonban változatlanul hagyja. A CLR törli a GOSUB és FOR utasításokhoz tartozó hivatkozásokat is, és végrehajt egy RESTORE utasítást!

Szintaxis: CLR

Példák:

```
(i)   CLR
(ii)  CLR: PRINT"A VALTOZOKAT TOROLTUK"
(iii) POKE 55,0: POKE 56,48: CLR: REM A MUNKATERULET VEGE $3000
(iv)  CLR: DIM AX(5,67)
```

A példák önmagukért beszélnek. Bármi is volt a változók értéke eddig, a programsor vagy a parancs végrehajtása után csak kezdőértékükkel használhatjuk őket. Ez számok esetén 0, sztringek esetén pedig az üres sztring (""). A (iii) példában a BASIC munkaterület végét jelző byte-okat átállítottuk. Ilyen esetben a CLR mindenképp szükséges, mert csak ezután veszi az interpreter figyelembe az új munkaterületet.

A (iv) példában az egyszer már deklarált AX tömböt akarjuk újra deklarálni. Ehhez azonban a változókat törölnünk kell. Ha ezt nem tesszük, akkor ?REDIM'D ARRAY hibaüzenetet kapunk.

Hibalehetőségek Az utasítás - helyes szintaxis esetén - mindig hiba nélkül végrehajtódik. Problémát csak az okozhat, hogy egyes változók értékei - amelyekre később szükség lenne - elvesznek. A CLR semlegesítése igen nehéz, ezért óvatosan bánjunk vele!

CMD

Rövidítés: cM *Token*:$9D(157)

Mód: mind parancs, mind program módban használható.

A CMD utasítás két funkciót egyesít:

1/ Segítségével módosíthatjuk az elsődleges output file-t;

2/ A CMD utasítást különböző értékek kiírására éppúgy használhatjuk, mint a PRINT utasítást. Az egyes értékeket elválasztó, illetve a sor végén szereplő ',' és ';' jeleknek ugyanaz a hatása.

Szintaxis: CMD <aritmetikai kifejezés> [<nyomtatási kép>]

Példák:

```
(i)    OPEN 5,4
       CMD 5: REM A TOVABBI OUTPUT A NYOMTATORA MEGY.
       CMD 5,: REM A TOVABBI OUTPUT A NYOMTATORA MEGY CRLF NELKUL
       CMD 5,"HELLO"
       PRINT=5:CMD PRINT : REM SZINTAKTIKUSAN HELYES

(ii)   OPEN 4,4 : CMD 4 : INPUT"NEV";K$

(iii)  10 OPEN 4,4: CMD 4
       20 PRINT "EREDMENYEK"
       30 GET K$:IF K$=""THEN GOTO 30
       40 PRINT X,Y,Z

(iv)   PRINT#4:CLOSE4:END
(v)    OPEN 1,4: CMD 1: LIST
       PRINT#1: CLOSE 1
```

Az (i) alatt felsorolt példák önmagukért beszélnek.

Ha a CMD 5, "HELLO" utasítást összehasonlítjuk a PRINT#5, "HELLO" utasítással, rögtön világossá válik, mennyire hasonlóak. Zavaró azonban, hogy míg a CMD 5 parancs hatására a nyomtató *'hallgató'* állapotba kerül, addig a PRINT#5 hatása ennek pont az ellenkezője; az output eszköz *megszűnik 'hallgató'* lenni. Ez jól látszik a nyomtató esetén. CMD utasítás után a 'READY.' nem jelenik meg a képernyőn. Ehhez még egy üres PRINT# utasítást is ki kell adni.

A (ii) példa a CMD egy igen szerencsétlen hatását mutatja be; az input promt a CMD-ben specifikált file-ba íródik, nem a képernyőre.

Hasonló a célja a (iii) példának is. A GET hatására az elsődleges output file újból a képernyő lesz, és az X,Y,Z értéke a *képernyőre* kerül. A (iv) példa mutatja, hogyan kell a CMD-ben használt file-t helyesen lezárni.

Megjegyezzük, hogy programhiba esetén a CMD hatása megszünik, és az elsődleges output egység újból a képernyő lesz. A fenti hatások miatt a CMD utasítást főleg listázásra célszerű használni úgy, ahogy azt az (v) példában látjuk. Eredmények kiírására a PRINT# utasítás használata egyszerűbb.

Hibalehetőségek A példák során már utaltunk a CMD utasítás néhány furcsaságára. Célszerű csak listázásra használni.

COLLECT

Rövidítés: colL *Token*:$F3(243)

Mód: mind parancs, mind program módban használható.

Újraszervezi a lemezegységen levő file-okhoz tartozó blokkokat.

Szintaxis: COLLECT [D<arit.kif.1>][{ , | ON } U<arit.kif.2>]

Az utasítás az <arit.kif.2> hardver egységszámú lemezegység <arit.kif.1> sorszámú meghajtójában levő lemez blokkjait szervezi újjá. Ha elhagyjuk a hardver egységszámot, akkor az utasítás a 8-as lemezegységre vonatkozik. Ha elhagyjuk a meghajtó számát, akkor a COLLECT mindig a 0-ás meghajtóra vonatkozik. Az utasítás a véletlen file-ok blokkjait törli. Ilyen lemezekre tehát ne alkalmazzuk az utasítást.

Példák:

```
(i)   COLLECT: PRINT DS$
(ii)  COLLECT D1
(iii) COLLECT D1 ON U10
```

Az utasítás (i) alatt használt formája a leggyakoribb. A géphez egyetlen egymeghajtós lemezegységet kapcsoltunk, aminek hardver száma 8. Az utasítás a lemez blokkjait újjászervezi.

A (ii) alatti formát abban az esetben használhajuk, ha a C-16-hoz egy 8-as hardver számú duál lemezegységet csatlakoztattunk. Ekkor a parancs az 1-es számú meghajtóban levő lemezt szervezi újjá.

A (iii) esetben gépünkhöz egy 10-es egységszámú duál lemezegységet csatlakoztattunk, s a parancs annak az 1-es meghajtójában levő lemez blokkjait szervezzi újjá.

Hibalehetőségek Ha a parancsban szereplő egységszámú lemezegység nincs a rendszerben, akkor ?DEVICE NOT PRESENT ERROR hibajelzést kapunk. Ha egymeghajtós lemmezegység esetén az 1-es meghajtóra hivatkozunk, akkor nem kapunk hibajelzést, csak a lemezegység lámpája kezd el villogni. A hibacsatorna kiolvasásakor ilyenkor?DRIVE NOT READY üzenetet kapunk (hibakódja 74). Ha az újjászervezés közben olvasási vagy íráshiba történik, az újjászervezés abbamarad, és a keletkezett hibáról a hibacsatorna olvasásával tájékozódhatunk.

COLOR

Rövidítés: cO *Token*:$E7(231)

Mód: mind parancs, mind program módban használható.

Az egyes színtípusok valódi színét állíthatjuk be. Az öt színtípus a következő:

színtípus	**jelentés**
0	háttér(*papír*) színe
1	írás(*tinta*) színe
2	több szín#1
3	több szín#2
4	keret színe

Szintaxis: COLOR <színtípus>,<szín> [<fényerő>]

A <színtípus> a 0-4 intervallumba kell hogy essen. A <szín> a lehetséges 16 színnek megfelelő 1-16 számérték bármelyike lehet. A <fényerő>-nek a 0-7 intervallumba kell esnie. 0 a legsötétebb, 7 a legvilágosabb.

Példák:

```
(i)   COLOR 0,3,0
(ii)  COLOR 1,I,7
```

Az (i) példa a háttér színét pirosra változtatja. A (ii) példa az írás színét az I. szín legvilágosabbjára változtatja.

Hibalehetőségek Az aritmetikai kifejezések kiértékelése közben számos hibalehetőség adódik. Ha ezek értéke nem esik a jelzett határok közé, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk. Nem megfelelően megválasztott színkombinációk esetén a képernyő olvashatatlanná válik.

CONT

Rövidítés: cO *Token:* $9A(154)

Mód: csak parancs módban használható.

A program továbbindítására szolgál, miután a program futása egy STOP vagy END végrehajtása, vagy a <STOP> billentyű megnyomása miatt megszakadt.

Szintaxis: CONT

A BASIC program futása közben a program végrehajtását ellenőrző ciklus beállít bizonyos mutatókat, amelyek segítségével az interpreter a CONT utasítást végrehajtja.

A CONT utasítás megléte természetesen lassítja a program futását. A program megállása után a **változó**kat a PRINT utasítással kiírathatjuk, **módosíthatjuk** értéküket, és a CONT még ezután is végrehajtható. Néhány esetben (pl. CLR, NEW, ?SYNTAX ERROR, illetve a program átszerkesztése után) a program futása már nem folytatható.

Hibalehetőségek A leírásban is jelzett esetekben a program futása nem folytatható. Ilyenkor ?CAN'T CONTINUE ERROR hibajelzést kapunk. (GOTO-val lehet kísérletezni.)

COPY

Rövidítés: coP *Token:*$F4(244)

Mód: mind parancs, mind program módban használható.

Lehetővé teszi egyes file-ok, vagy a lemez valamennyi file-jának átmásolását.

Szintaxis:

COPY [D<arit.kif.1>,] <sztring.1> TO [D<arit.kif.2>,] <sztring.2> [{ , | ON } U<arit.kif.3>]

Az <arit.kif.1> és <arit.kif.2> értéke a meghajtó számát adja meg. Értékük 0, vagy 1 lehet. Ha elmaradnak, azt az interpreter 0-nak tekinti. <arit.kif.3> a lemezegység hardver számát definiálja. Ha nem adjuk meg, az 8-nak számít. <sztring.1> **a** másolandó file neve, <sztring.2> pedig a másolati példány neve.

Példák:

```
(i)     COPY D0, "REGI" TO D1, "UJ"
        COPY D0, "REGI" TO "UJ"
        COPY D0, "PROGRAM" TO D1, "PRG" ON U9
```

Ezek a példák önmagukért beszélnek. A második sorban álló COPY parancs mutatja azt a lehetőséget, hogy egy file-nak ugyanazon a lemezen is létrehozhatjuk - természetesen más néven - **a** másolatát.

```
(ii)    COPY D0 TO D1
(iii)   COPY D0, "PROGRAM.*" TO D1, "*"
        COPY D0, "???.PRG" TO D1, "*"
```

A (ii) **alatti** parancs a COPY egy **speciális** használatát mutatja. Ha hiányoznak a file-nevek, akkor a lemezegység **valamennyi file-t egyenként** átmásolja a másik meghajtóban levő lemezre. Ha ezen már voltak file-ok, azok rajta maradnak a lemezen.

A (iii) példa azt mutatja, hogy a file-nevekben szereplő speciális karakterek hatása a COPY utasításnál is érvényesül. Az első sorban levő COPY **valamennyi** "PROGRAM." sztringgel kezdődő nevü file-t - ugyanazzal a névvel - átmásol. Ilyen esetben az új névnek megfelelő sztring egyetlen "*" karakterből kell hogy álljon. A második COPY parancs a D0-ás meghajtóban levő lemezen levő összes olyan file-t átmásolja - változatlan néven - melyek neve pontosan 7 karakter hosszú és az utolsó négy karakter ".PRG".

Hibalehetőségek Az aritmetikai és sztringkifejezések

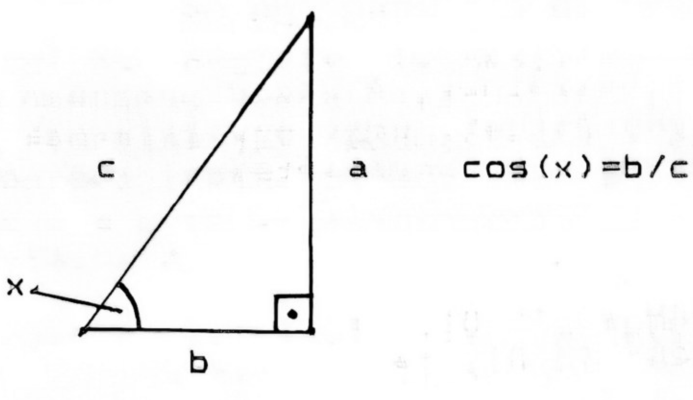
kiértékelése közben számos hiba adódhat. Ha az aritmetikai kifejezések értéke nem megfelelő, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk. Ha a másolás közben olvasási, vagy íráshiba történik a másolás félbeszakad, és a hibának megfelelő üzenetet kapunk. Ha a hiba jellege olyan, akkor előfordulhat, hogy csak a lemezegység hibalámpája jelez. Ilyenkor a hibacsatorna olvasásával (?DS$) kaphatjuk meg a hibaüzenetet. Ugyanez a helyzet, ha a másolandó file nincs a lemezen, vagy egy olyan néven akarjuk tárolni a másolatot, amilyen nevű file már létezik a lemezen.

COS

Rövidítés: nincs *Token*: $BE(190)

Mód: mind parancs, mind program módban használható.

A radiánban megadott szög koszinuszát számítja ki. Az alábbi ábra x és cos x viszonyát mutatja:



Szintaxis: COS (<aritmetikai kifejezés>)

Példák:

```
(i)   PRINT COS(1.25)
(ii)  PRINT COS(90*<pi>/180): REM =1
(iii) 10 DO: INPUT X
      20 PRINT COS(X)^2+SIN(X)^2
      30 LOOP
(iv)  X=ALFA-SIN(ALFA):Y=1-COS(ALFA)
```

Az első két példa önmagáért beszél. A (iii) példa a **Pitagorasz-tételt** 'ellenőrzi'. Az X értékétől függetlenül mindig nagy pontossággal 1-et kell a programnak nyomtatnia. A program futása a <STOP> lenyomásával állítható csak meg.

Az utolsó példákban speciális trigonometrikus függvényeket számitunk ki. A (iv) például egy **ciklois** X,Y koordinátáit adja meg.

Hibalehetőségek Az aritmetikai kifejezés kiértékelése közben többfajta hibajelzést is kaphatunk. Ha a kiértékelés sikerrel befejeződött, maga a COS kiszámítása már hiba nélkül megtörténik.

DATA

Rövidités: dA *Token:*$83(131)

Mód: csak program módban használható.

A fenti utasitás lehetővé teszi, hogy a programban számokat, sztringeket tároljunk, s ezeket a READ utasitás segitségével beolvassuk. A READ utasitások abban a sorrendben olvassák be ezeket az adatokat, ahogy azok a DATA utasitásokban egymást követik.

Szintaxis: DATA <konstanslista>

A konstanslista elemei egymástól vesszővel (,) elválasztott konstansok. Mind szöveg- (sztring-), mind számkonstansok szerepelhetnek a listában. A szövegkonstansokat nem kell idézőjelek közé tenni, kivéve ha tartalmaznak vesszőt (,) vagy grafikus jeleket.

Példák:

```
(i)   151 DATA"KOVACS PETER,1957","NAGY ISTVAN,1964"
      152 DATA"ERDOS ILONA,1965"
(ii)  1055 DATA GEPI KOD,120,169,46,133,96
(iii) 2000 DATA 1,1,2,2,3,0: REM MUVELETEK RANGJA
      2010 DATA 0,1,2,3,4,5,6,7,8,9: REM ERVENYES SZAMJEGYEK
      2020 DATA ,+-,*,/,^: REM MUVELETI JELEK
```

Az első példa szövegkonstansok DATA-ban való tárolását mutatja. A sztringekben szereplő vesszők miatt az idézőjeleket nem hagyhatjuk el. A FOR I = 1 TO 3 : READ X$: ? X$: NEXT utasitással mind a három szövegkonstans kiiratható. A második példa bemutatja, hogyan lehet bizonyos adatokat megtalálni a DATA-k közt. Addig hajtjuk végre a READ X$ utasitást, amig X$="GEPI KOD" nem teljesül. A "GEPI KOD" sztringben nem szerepel vessző, ezért az idézőjeleket elhagyhattuk. A (iii) példa mutatja, hogy a DATA-kban célszerű strukturáltan elhelyezni az

adatokat, hogy könnyebben cserélhetők legyenek.

A DATA utasítás végrehajtása az utasítás kihagyását jelenti, azzal a különbséggel, hogy nem az egész programsor marad ki (mint a REM esetén), hanem csak az utasítás. A rutin megkeresi a következő kettőspontot vagy a programsor végét, és onnan folytatja a program végrehajtását. Tekintsük a következő példát:

```
10 X=2
20 DATA 3: X=5
30 PRINT X
```

A program a 20. sorban átlépi a DATA-t és az X=5 utasítást hajtja végre. Ennek megfelelően a 30. sorban az 5 íródik ki.

Hibalehetőségek A READ utasítás néhány rutinja megegyezik az INPUT utasítás hasonló rutinjaival. Emiatt a nem siftelt, sztring elején álló szóközök és bizonyos grafikus jelek elvesznek. ?SYNTAX ERROR hibaüzenet egy DATA utasítással kapcsolatban általában a hozzá tartozó READ utasítás hibáját jelenti. READ X$ sohasem okozhat problémát (kivéve, ha nincs több adat). READ X esetén nem mindegy, hogy mi a következő adat. Felesleges vessző is okozhat problémát. A DATA 1,2,3, utasítás négy konstanst tartalmaz, a negyedik egy üres sztring ("").

DEC

Rövidítés: nincs *Token*:$D1(209)

Mód: mind parancs, mind program módban használható.

A függvény az argumentumaként szereplő sztringet hexadecimális számnak tekinti, és annak decimális értékével tér vissza.

Szintaxis: DEC(<sztring.kif.>)

A sztring maximum 4 hexadecimális számjegyet(0-9,A-F) tartalmazhat.

Példák:

```
(i)     PRINT DEC("AF") : REM =175
        PRINT DEC("1000") : REM =4096
(ii)    10 DO WHILE X<256
        20 PRINT X,HEX$(X),DEC(HEX$(X)
        30 LOOP
(iii)   210 DO UNTIL C$=CHR$(13)
        220 INPUT A$,B$: GETKEY C$
```

```
230 PRINT HEX$(DEC(A$)+DEC(B$))
240 GETKEY C$
250 LOOP
```

Az (i) alatti két példa a DEC közvetlen hatását mutatja be. A (ii) alatti rövid program azt mutatja be, hogy a DEC függvény a HEX$ függvény inverze. A program az első 255 számot írja ki decimális, hexadecimális, majd ismét decimális alakban (20. sor).

A (iii) alatti példa segítségével hexadecimális számok összeadását gyakorolhatjuk. A program indítás után két hexadecimális számot vár inputként. Ha ezeket a számokat fejben (vagy papíron) összeadtuk, nyomjunk meg egy tetszőleges billentyűt, a gép kiírja a választ. Ha ezután a <RETURN> billentyűt nyomjuk meg, a program futása abbamarad. Bármely más billentyű megnyomása esetén újabb két szám összeadását végeztethetjük el.

Hibalehetőségek A sztring kifejezés kiértékelése közben számos hibalehetőség adódhat. Ha a sztring értéke nem esik a $0000-$FFFF intervallumba, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

DEF FN

Rövidítés: dE (FN-nek nincs rövidítése) *Token:* DEF $96(150)
FN $A5(165)

Mód: csak program módban használható.

A DEF FN utasítás segítségével egyváltozós aritmetikai függvényeket definiálhatunk, amelyekre azután az FN segítségével hivatkozhatunk. A definícióban szerepel a függvény neve és a változója.

Szintaxis: DEF FN <változó> (<változó>) = <aritmetikai kif.>

Az első <változó> a **függvény neve**, a második a **változója**. A függvényt definiáló aritmetikai kifejezés legfeljebb egy BASIC-sornyi lehet. A függvény definiálása után az FN <változó> (<aritmetikai kif.>) alakban használhatjuk a függvényt. Mindkét változó csak *valós* lehet.

Példák:

(i)
```
10 A=3: B=4: C=-5
20 DEF FN F(X)=A*X*X+B*X+C
```

```
    30 FOR I=0 TO 2 STEP .1
    40 PRINT X, FN F(I)
    50 NEXT I

(ii)  300 DEF FN DEEK(X)=PEEK(X)+256*PEEK8X+1)
(iii) 400 DEF FN MIN(X)=(A+B)/2-ABS(A-B)/2
      410 DEF FN MIN(X)=-(A>B)*B-(B>=A)*A
```

Első példánk egy másodfokú függvényt definiál, majd annak helyettesitési értékeit számitja ki a [0,2] intervallumban.

(ii) egy '**dupla pontosságú** PEEK'-et definiál. Ha két egymás utáni byte egy cim alsó illetve felső byte-ját tárolja (ebben a sorrendben), akkor az FN DEEK(X) függvény ennek a cimnek az értékét számitja ki.

A harmadik példában két megoldást láthatunk a ***minimum*** *függvény* kiszámitására. Ez nem igazi függvény, hiszen az X csak ál-változó (dummy variable), a szintaxis kedvéért szerepel. Az A és B értéket a függvény kiszámitása előtt meg kell adni:

```
A=12: B=24: PRINT FN MIN(X)
```

Az X változó helyén tetszőleges aritmetikai kifejezés szerepelhet: FN MIN (0), FN MIN (X*Y-10) értéke megegyezik.
A függvénydefiniciók további függvénydefiniciókat is tartalmazhatnak. Például

```
123 DEF FN EX(X)=1+X+X^2/2+X^3/2/3+FN E(X)
124 DEF FN E(X)=X^4/2/3/4+X^5/2/3/4/5+X^6/1/2/3/4/5/6
```

megengedett.

A függvények hasonlóan más változókhoz újra definiálhatók

```
175 DEF FN Y(X)=Y: DEF FN Y(X)=X
```

helyes.

Megjegyezzük, hogy a függvény kiértékelése közben a függvény definiciójában szereplő változó értéke nem változik meg:

```
10 X=77
20 DEF FN Q(X)=X*X-5*X+6
30 PRINT FN Q(3),X
```

A 30. sorban X értékeként nem 3, hanem 77 iródik ki. A függvény kiértékelésekor X-et ideiglenesen tárolja az interpreter, majd visszaállitja az eredeti értékét.

Függvényeket csak program futása közben definiálhatunk. A már

definiált függvények azonban a program megállása után parancs módban is használhatók. Irjuk be és futtassuk le a következő egysoros programot:

```
10 DEF FN CS(X)=COS(X*pi/180)
```

Ezt követően a CS függvény segítségével fokokban adott szög koszinuszát is kiszámíthatjuk: PRINT FN CS(90) például 1-et ad eredményül.

Hibalehetőségek Ha a DEF után az egyenlőségig valamilyen hibát követünk el, akkor ?SYNTAX ERROR hibajelzést kapunk. Ha az aritmetikai kifejezés kiértékelése közben történik hiba, akkor a hibajelzést a megfelelő FN-t tartalmazó sorban kapjuk. Ha az FN utasítást a hozzá tartozó DEF FN utasítás előtt hívjuk, akkor ?UNDEF'D FUNCTION ERROR hibajelzést kapunk. (A CLR a függvénydefiníciókat is törli!)

DELETE

Rövidítés: deL *Token*:$F7(247)

Mód: csak parancs módban használható.

Több programsort töröl egyszerre.

Szintaxis: DELETE <sorszám.1> [- [<sorszám.2>]]

<sorszám.1> az első **már**, <sorszám.2> pedig az utolsó **még** törlendő sor száma. A sorszámok csak előjel nélküli egész számok lehetnek

Példák:

```
(i)    DELETE 168
       DELETE 1000-
       DELETE 1000-2400
       DELETE -300
```

A példák önmagukért beszélnek. A paraméterek használata megegyezik a LIST parancséval. Ha megadjuk a törlendő **első**, illetve utolsó sor számát, akkor ezek a sorok is törlődnek. A harmadik sorban szereplő DELETE parancs tehát az 1000., illetve a 2400. sort is törli (és az összes köztük levőt).

Hibalehetőségek Nincs, de ha nem vigyázunk, nem kívánt sorokat is törölhetünk. Mint minden *törlő* jellegű paranccsal, ezzel is óvatosan kell bánni.

DIM

Rövidítés: dI *Token:*$86(134)

Mód: mind parancs, mind program módban használható.

Az utasítás a DIM-et követően megadott nevű, típusú, dimenziójú és méretű tömbelemeknek helyet foglal a memóriában. A tömb neve, típusa tetszőleges lehet, az egyes indexek értéke 0-tól a DIM utasításban megadottig terjedhet, de legfeljebb 32767 lehet.

Szintaxis: DIM <tömbváltozó lista>

A lista egymástól vesszővel elválasztott indexes változókat tartalmaz.

Példák:

```
(i)    10 DIM A%(12),B1%(100,12),B(210),KJ$(320)
(ii)   410 INPUT "N=";N: DIM TOMB(N*N+1)
(iii)  3000 FOR J=1 TO 3: READ A$: MES$(J)=A$:NEXT J
(iv)   DIM XF(23)
```

A DIM utasítás használata egyszerű. A problémát inkább az okozza, hogy nagyobb adatmennyiséget a programjaink által kezelhető módon már nehéz megszervezni. A tömbök **dinamikusan definiálhatók**; erre példa a fenti (ii) példa, amiben a TOMB dimenzióját az N*N+1 kifejezés adja meg. Az (iii) példa **implicit tömbdefiníciót** tartalmaz. Ez azt jelenti, hogy tömbváltozókat a tömbök definiálása nélkül is lehet használni, feltéve, hogy indexeik nem haladják meg a 10-et. A MES$(1) kiértékelésekor a memóriában a tömb elhelyezésére is sor kerül. A hatás ekvivalens a DIM MES$(10) utasítással.

A legkisebb index értéke 0 lehet. Több számítógép nem teszi lehetővé ennek az indexnek a használatát. Ugyanakkor egy tömb 0. elemét különféle számításokban jól fel lehet használni. Például a

```
10 DIM A(20): FOR X=1 TO 20
20 INPUT N: A(X)=N: A(0)=A(0)+N: NEXT
```

programrész 20 elemet olvas be az A(X) tömbbe, összegüket pedig elhelyezi az A(0) tömbelemben.

A CLR utasítás mind az egyszerű, mind a tömbváltozókat törli. A tömbváltozók **külön törlése** a következő paranccsal (ami programból is használható) elvégezhető:

```
POKE 49,PEEK(47): POKE 50,PEEK(48)
```

A 3.2 paragrafusban részletesen leírtuk, hogy egy tömb elemei hogyan helyezkednek el a memóriában. A következő BASIC paranccsal kiírathatjuk a tömb által foglalt helyet:

```
F=FRE(0): DIM S$(14,20): PRINT F-FRE(0)
```

Hibalehetőségek Hibát elsősorban az indexhatárok kiértékelése közben kapunk. ?OUT OF MEMORY ERROR hibajelzést kapunk, ha nincs elegendő hely a tömb elhelyezésére. Ugyanazt a tömböt csak egyszer definiálhatjuk egy programban. A második kísérletre ?REDIM'D ARRAY ERROR hibajelzést kapunk. Ugyanezt a hibajelzést kapjuk abban az esetben is, ha implicit módon definiáltuk a tömböt, s azt követően újra definiáltuk:

```
X$(0)="ABCD":DIM X$(22)
```

A leggyakoribb hiba, ha szintaktikus vagy tervezési hiba miatt egy tömbváltozó indexei nem esnek a megfelelő intervallumba. Ilyenkor ?BAD SUBSCRIPT ERROR hibaüzenetet kapunk.

DIRECTORY

Rövidítés: diR *Token*:$EE(238)

A lemezegység katalógusát a képernyőre listázza.

Szintaxis:

DIRECTORY [D<arit.kif.1>] [{ , | ON } U<arit.kif.2>] [,<sztring kif.>]

Az első aritmetikai kifejezés a meghajtó sorszámát adja meg. értéke csak 0, vagy 1 lehet. A második aritmetikai kifejezés a lemezegység hardver száma. A sztring kifejezés a listázandó file-ok nevét adja meg. Ha elmarad, valamennyi file-t kilistázzuk.

Példák:

```
(i)   DIRECTORY
      DIRECTORY D1 ON U9, "BASIC*"
```

Az első sor a 8-as lemezegység 0-ás meghajtójában levő lemez katalógusát listázza a képernyőre. A második sorban álló parancs hatására a 9-es lemezegység 1-es meghajtójában levő lemez "BASIC"-kel kezdődő nevű file-jai kerülnek csak kilistázásra.

Hibalehetőségek Az aritmetikai és sztringkifejezések kiértékelése közben számos hiba történhet. Ha a lemezegység jelez hibát, azt csak a hibacsatorna kiolvasásával kapjuk meg.

DLOAD

Rövidítés: dL *Token:*$F0(240)

Mód: mind parancs, mind program módban használható.

A lemezegység adott file-ját tölti be a memóriába.

Szintaxis:
DLOAD <sztring kif.> [D<arit.kif.1>] [{, | ON} U <arit.kif.2>]

A <sztring kif.> a betöltendő file nevét, <arit.kif.1> a meghajtó sorszámát, <arit.kif.2> pedig a lemezegység hardver számát jelenti. Ha a file neve ? vagy * karaktert tartalmaz, akkor az első - a file névnek megfelelő - file-t tölti be az interpreter.

Példák:

```
(i)     DLOAD "BEMUTATO
        DLOAD "BEMUTATO",D1
(ii)    A$="BEMUTATO": DLOAD (A$),D1
(iii)   DLOAD "PROGRAM",D0 ON U9
```

Az (i) alatt felsorolt két parancs a DLOAD leggyakoribb használatát mutatja be. Az első esetben a "BEMUTATO" nevű program a 8-as lemezegység 0-ás meghajtójában levő lemezről, a második esetben ugyanezen lemezegység 1-es meghajtójáról töltődik be.

A (ii) alatti példa azt mutatja, hogy a DLOAD paraméterei közt nem csak sztring konstansok szerepelhetnek. Ebben az esetben azonban a kifejezést kerek zárójelek közé kell tenni.

A (iii) példában a töltés a 9-es lemezegység 0-ás meghajtójáról történik. A D0 elhagyható lenne.

Hibalehetőségek Az aritmetikai és sztring kifejezések kiértékelése közben számos hiba történhet. Ha a meghajtó sorszáma és a lemezegység hardver száma nem esik a lehetséges értékek intervallumába, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk. Ha a szóban forgó lemezegység nincs a rendszerben, akkor ?DEVICE NOT PRESENT ERROR hibajelzést kapunk. Ha az adott nevű file nincs a lemezen, akkor ?FILE NOT FOUND ERROR hibajelzést kapunk.

A töltés közben előforduló hibákról általában nem kapunk jelzést, csak a lemezegység lámpája jelzi a hibát. A hiba természetéről a lemez hibacsatornájának kiolvasásával kapunk információt.

DO

Rövidítés: nincs *Token*: $EB(235)

Mód: mind parancs, mind program módban használható.

Lehetővé teszi a DO és a LOOP utasítások közti programrész ismételt végrehajtását. A **kilépés feltételét** a DO-t, vagy a LOOP-ot követő WHILE és UNTIL utasításokban adhatjuk meg. A ciklusból az EXIT utasítás segítségével bármikor kiléphetünk.

Szintaxis:

```
          ( UNTIL )
    DO [ <         > <logikai.kif.> ]
          ( WHILE )
```

A DO és a hozzátartozó LOOP utasítás közti programrészt **ciklusmagnak** hívjuk. A DO vagy a LOOP utasításban szereplő UNTIL, illetve WHILE utasításokat követő logikai kifejezéseket a **kilépés feltételének** nevezzük. Ezeknek kell a WHILE esetében hamissá, az UNTIL esetében igazzá válni ahhoz, hogy a ciklus *lejárjon*. A ciklusmagban elhelyezett EXIT utasításokat *kilépési pontoknak* hívjuk. Egy ilyen utasítás végrehajtása a ciklusból való azonnali kilépést eredményez.

Példák:

```
(i)    10 DO WHILE A$="": GET A$: LOOP
       10 DO UNTIL A$<>"": GET A$: LOOP

(iia) FOR I=KE TO VE STEP LE
       ...
       NEXT I
(iib) I=KE: DO
       ...
       I=I+LE
       LOOP WHILE (LE<0 AND I>=VE) OR (LE>=0 AND I<=VE)
(iic) I=KE: DO WHILE (LE<0 AND I>=VE) OR (LE>=0 AND I<=VE)
       ...
       I=I+LE
       LOOP
```

Az (i) példa a DO egy tipikus használatát mutatja be. A program végrehajtása addig áll egy helyben, míg meg nem nyomunk egy billentyűt. A két 10. sor csak az UNTIL/WHILE megválasztásában tér el egymástól. Az első esetben a ciklus csak akkor ismétlődik, ha a WHILE-t követő logikai feltétel igaz, azaz A$ egy üres sztringet tartalmaz. A második esetben a ciklus akkor ismétlődik csak, ha az UNTILT követő feltétel nem igaz, azaz A$<>"" hamis. Mindkettő ekvivalens a 10 GETKEY A$ programsorral.

A (ii) alatti példa egy FOR ciklust tartalmaz. A (iia) alatti példa ennek DO ciklussal való helyettesítését mutatja be. A ciklusból való kilépést ellenőrző WHILE azért olyan bonyolult, mert nem ismerjük LE előjelét. Annak ismeretében az OR valamelyik tagját elég beírni. Ha például LE>0, akkor a LOOP így írható:

```
LOOP WHILE I<=VE
```

A (iic) egyetlen apróságban tér el a FOR ciklustól. A FOR ciklus legalább egyszer mindig lefut. A WHILE előrehozása a DO-ba azt eredményezi, hogy KE>VE esetén a ciklus egyetlen egyszer sem fut le. Bizonyos esetekben ez nagyon hasznos lehet.

A FOR átírása is mutatja, hogy a DO ciklust elsősorban olyan esetekben használjuk, amikor *ciklusváltozóra* nincs szükség, vagy a *ciklusváltozók* közül nem tudunk egy meghatározót kiválasztani. A DO ciklus lényegesen gyorsabban fut, mintha IF...THEN GOTO... alakú struktúrával valósítanánk meg.

A DO utasítások tetszőleges mélységben egymásba skatulyázhatók:

```
DO
 ...
 DO
  ...
 LOOP UNTIL ...
 ...
LOOP WHILE ...
```

A DO utasítás végrehajtása az azt esetleg követő kilépési feltétel kiértékelésével kezdődik. Ha ez alapján ki kell lépni a ciklusból, akkor az interpreter a DO utasítást követően elkezdi az első LOOP utasítást megkeresni, s ha megtalálta, az azt követő első utasításra adja a vezérlést. Ha még nem járt le a ciklus, akkor a visszatérést biztosító információk a verembe kerülnek, s az interpreter rátér a DO-t követő első utasítás végrehajtására.

Hibalehetőségek A DO-t esetleg követő kilépési feltétel kiértékelése közben számos hiba adódhat. A FOR, GOSUB is használja a vermet, így az egyes struktúrák egymásba skatulyázásának határt szab a verem nagysága. Ha már nincs elég hely a veremben, akkor ?OUT OF MEMORY ERROR hibajelzést kapunk. Ha a ciklus lejárt, és az interpreter nem találta meg a megfelelő LOOP utasítást, akkor ?LOOP NOT FOUND ERROR hibajelzést kapunk. Rosszul strukturált programok gyakran adják ezt a hibajelzést.

DRAW

Rövidítés: dR *Token:*$E5(229)

Mód: mind parancs, mind program módban használható.

A nagyfelbontású képernyőre rajzol pontokat, szakaszokat.

Szintaxis:
DRAW [<színtípus>] , [<x1>,<y1>] [TO <x2>,<y2>]

A rajzolás közben bekapcsolódó pontok színe <színtípus> színű lesz. Ha a paraméter elmarad, akkor az interpreter az 1-es színtípust (írás színe) használja. értékének a kiválasztott grafikus móddal összhangban kell lennie. <x1>,<y1> a kezdőpont koordinátái. Ha elmarad, akkor a kezdőpont a grafikus kurzor pillanatnyi helyzete lesz. A TO-val bevezetett részek a szakasz végpontjának koordinátáit adják meg. **Mind a kezdőpont, mind a TO-s rész tetszés szerinti sokszor ismételhető.** Igy egyetlen DRAW utasítással több töröttvonal is rajzolható.

Példák:

```
(i)     DRAW 1,100,160
(ii)    DRAW 0,100,160
(iii)   DRAW 1,0,0,319,199
(iv)    DRAW 1,100,100 TO 150,100 TO 150,150 TO 100,150

(v)      50 REM RESZEG TENGERESZ
        100 GRAPHIC 1,1: LOCATE 160,100
        150 DO
        200 X=319*RND(0): Y=199*RND(0)
        250 DRAW ,TO X,Y
        300 GET A$
        350 LOOP WHILE A$=""
        400 GETKEY A$
        450 GRAPHIC 0
```

Az (i) alatti példa a nagyfelbontású képernyőre rajzol egyetlen pontot. A parancs kiadása előtt ki kell adni a megfelelő GRAPHIC parancsot, különben hibajelzést kapunk. A (ii) példa ugyanezt a pontot törli. A (iii) példa a nagyfelbontású képernyőn keresztben meghúzza az egyik átlót.

A (iv) alatti példában a DRAW paramétereinek ismételhetőségét használjuk fel egy négyzet megrajzolására.

(v)-ben egy részeg tengerész bolyong fel-alá a képernyőn, útjának megrajzolására használjuk a DRAW utasítást.

Hibalehetőségek Az aritmetikai kifejezések kiértékelése közben számos hiba történhet. Ha a <színtípus> értéke nem felel meg a használt grafikus módnak, akkor ?ILLEGAL QUALITY ERROR hibajelzést kapunk. A paraméterek elhagyása esetén ügyelni kell a vesszők megfelelő elhelyezésére. Ha nem adtuk még ki a megfelelő GRAPHIC utasítást, akkor ?NO GRAPHICS AREA ERROR hibajelzést kapunk.

DS és DS$

Fenntartott BASIC változók

A DS valós, illetve a DS$ sztring változókat a C-16 speciálisan kezeli. Bármelyikükre való hivatkozás a lemezegység hibacsatornájának olvasását eredményezi. DS-ben a hiba kódja, DS$-ban pedig a teljes üzenetet kapjuk vissza. A rendszer mindig az utoljára használt lemezegység hibacsatornáját olvassa, kezdetben a 8-as egységszámúét. A hibaüzenetekről, a hibacsatornáról részletesen az 5. fejezetben lesz szó.

Példák:

```
(i)    BACKUP D0 TO D1: ? DS$
(ii)   SCRATCH "PROBA*": ? DS$
```

Mindkét példa azt szemlélteti, hogy általában minden egyes lemezművelet után célszerű a hibacsatornát kiolvasni. Ez program módban természetes, hiszen ilyenkor a lemezegység lámpájára sem hagyatkozhatunk. (i) alatt egy teljes lemez másolása után olvassuk ki a hibacsatornát és íratjuk ki a képernyőre az üzenetet.

A (ii) példában a "PROBA" névvel kezdődő valamennyi file-t töröljük a lemezről. Ilyenkor mindenképpen célszerű a hibacsatorna kiolvasása, hogy megtudjuk, pontosan hány file-t is töröltünk.

Hibalehetőség Nincs.

DSAVE

Rövidítés: dS *Token*:$EF(239)

Mód: mind parancs, mind program módban használható.

A memóriában tárolt BASIC program lemezre történő kiírását (mentését) végzi.

Szintaxis:

DSAVE <sztring kif.> [,D<arit.kif.1>] [{ , | ON } U<arit.kif.2>]

A <sztring kif.> annak a file-nak a neve, ahová a memóriában levő programot menteni kell. Abban az esetben, ha egy már meglevő file tartalmát akarjuk felülírni, akkor a filenévnek a @ jellel kell kezdődnie. A két aritmetikai kifejezés értéke a meghajtó, illetve a lemezegység hardver számát adja meg.

Példák:

```
(i)     DSAVE "SZAMLAZO
(ii)    DSAVE "@SZAMLAZO
(iii)   DSAVE "RUTINOK",U9
(iv)    DSAVE "PROGRAMOK",D1,U9
```

Az (i) példa a DSAVE egyik leggyakoribb használatát mutatja. A memóriában levő BASIC program teljes egészében a "SZAMLAZO" nevü lemezes file-ba kerül. (A záró " jelet nem is kell kiírni.) A lemezegység aktivizálásakor a képernyőn a

SAVING "0:SZAMLAZO"

felirat jelenik meg. A *READY.* üzenet megjelenése jelzi, hogy a kiírás befejeződött.

(ii)-ben a memóriában levő programot a már meglevő "SZAMLAZO" nevű file-ba írjuk ki. A file eredeti tartalma természetesen elvész. Az utasítás akkor is korrektül végrehajtódik, ha a "SZAMLAZO" nevű file még nem létezik a lemezen.

A (iii)-as példa a memória tartalmát a 9-es lemezegység 0-ás meghajtójában levő lemezre menti ki "RUTINOK" file név alatt. Ez a file még nem létezhet a lemezen.

A (iv) példa a D paraméter használatát mutatja. A memória tartalma a 9-es lemezegység 1-es meghajtójában levő lemezre íródik ki.

Hibalehetőségek A kifejezések kiértékelése közben számos hiba adódhat. Ha a <sztring kif.> nem sztring konstans, akkor kerek zárójelek közé kell tenni:

```
A$="PROGRAM": DSAVE (A$)
```

Ellenkező esetben ?SYNTAX ERROR hibaüzenetet kapunk.

Ha a használni kívánt egység nincs a rendszerben, akkor ?DEVICE NOT PRESENT ERROR hibajelzést kapunk. Ha az adott nevű file már létezik, s a név előtt nem szerepel a @ karakter, akkor ?FILE EXISTS ERROR hibaüzenetet kapunk. Ha a lemezre írás közben íráshiba történik, a mentés azonnal befejeződik. A hibát ilyenkor csak a lemezegység lámpája jelzi. A hibacsatorna kiolvasásával megtudhatjuk a hiba pontos okát is.

EL és ER

Fenntartott BASIC változók

Mind a két változó a C-16 hibakezelő rendszeréhez tartozik. Ha működése közben az interpreter hibát észlel, a vezérlés a hibakezelő rutinra adódik át. A hibakezelő rutin a BASIC program része is lehet. Ebben az esetben EL a hibát közvetlenül okozó BASIC sor számát, míg ER a hiba kódját tartalmazza. ER értéke az 1-36 intervallumba esik. Ha nem történt hiba, akkor EL=65535, ER=-1.

Példák:

(i)
```
10 TRAP 2000
20 ?0/0
30 ?0/1
40 END
2000 PRINT "HIBA TORTENT!!!"
2010 PRINT "HIBA KODJA=";ER
2020 PRINT "HIBAS SOR SORSZAMA=";EL
2040 RESUME NEXT
```

Példánk egy hibakezelő rutint mutat be, amelyik nem tesz egyebet, csak kiírja a hiba kódját és helyét. A program futtatásakor a 20. sorban hiba történik, ezért a 2000-es hibakezelő rutinra adódik a vezérlés. A hiba kódja 20 lesz (?DIVISION BY ZERO ERROR), s a hibás sor száma ugyancsak 20. A 2040-es sor elérése után a program futása a 30. sornál folytatódik.

Hibalehetőségek Nincs.

END

Rövidítés: eN *Token:*$80(128)

Mód: mind parancs, mind program módban használható.

Az END végrehajtása a program futásának azonnali befejezését jelenti; az interpreter a 'READY.' üzenet kiírásával visszaadja a vezérlést a képernyő szerkesztőnek. A CONT parancs kiadásával folytathatjuk a program futását.

Szintaxis: END
Az utasításnak nincsenek paraméterei.

Példák:

```
(i)   100 PRINT#4: CLOSE 4: GOSUB 200: END
(ii)  130 IF SAV<>TR THEN PRINT "** NEM JO SAVSZAM **":END
(iii) 2000 GOSUB 1000:END:GOSUB 2000:END
(iv)  59999 END
      60000 <utasítások>
```

Négy példánkban az END különböző célú alkalmazásait igyekeztünk bemutatni. Az első példa a nyomtató lezárása után meghívja a 200 alatti alprogramot, majd befejezi a program futását. A második példában az END egy hibaág végén jelzi az interpreternek, hogy a program futását fel kell függeszteni.

A harmadik példa nem egy kész program részlete. A program ellenőrzésére két töréspontot is elhelyeztünk. A program megállása, a változók ellenőrzése után a CONT paranccsal folytathatjuk a program futását. Az utolsó példában a 60000 sorszámmal kezdődő alprogramok elé egy END utasítást tettünk, nehogy 'rácsorogjon' a program vezérlése az alprogramokra.

Hibalehetőség Nincs. (Leszámítva azt az esetet, hogy vezérlési hiba folytán rossz helyen áll meg a program.) Az END végrehajtásakor az interpreter nem írja ki a BREAK IN ... üzenetet.

ERR$

Rövidítés: eR (a $ már nem kell!) *Token:*$D3(211)

Mód: mind parancs, mind program módban használható.

Sztringfüggvény, amelyik az argumentumaként megadott sorszámú hiba szövegével tér vissza.

Szintaxis: ERR$(<arit.kif.>)

Az aritmetikai kifejezés értékének az 1-36 intervallumba kell esnie.

Példák:

```
(i)     ? ERR$(2) : REM = "FILE OPEN"
(ii)    10 FOR I=1 TO 36
        20 PRINT I;". HIBA=";ERR$(I)
        30 NEXT I
```

Az első példa eredményeként a megjegyzésben feltüntetett hibaüzenetet kapjuk. A (ii) példa kiírja a képernyőre az összes hibaüzenetet a hiba számkódjával együtt.

Hibalehetőségek Az aritmetikai kifejezés kiértékelése közben számos hibalehetőség adódik. Ha ennek értéke nem esik az 0-36 intervallumba, akkor ?ILLEGAL QUNTITY ERROR hibajelzést kapunk.

EXIT

Rövidítés: exI *Token:*$ED(237)

Mód: mind parancs, mind program módban használható.

DO...LOOP ciklusokban kilépési pontok elhelyezésére szolgál.

Szintaxis: EXIT

Az EXIT a ciklusból feltétel nélküli kilépést eredményez. Az interpreter megkeresi az EXIT-et követő első LOOP-ot, s az azt követő első utasításra adja át a vezérlést.

Példák:

(i)
```
10 INPUT A%,B%
20 DO
30 M%=B%-INT(B%/A%)*A%
40 IF M%=0 THEN EXIT
50 B%=A%: A%=M%
60 LOOP
70 ? "LNKO=";A%
```

Az (i) a LOOP-nál látható példa módosítása. Az A%,B% legnagyobb közös osztóját számítja ki, de úgy szerveztük a számítást, hogy a kilépésre a ciklus közepén kerüljön sor.

Hibalehetőségek Ha az interpreter nem találja meg a LOOP utasítást, akkor ?LOOP NOT FOUND ERROR hibaüzenetet kapunk. Ha LOOP-hoz nem találja meg a megfelelő DO utasítást, akkor ?LOOP WITHOUT DO ERROR hibajelzést kapunk.

EXP

Rövidítés: eX *Token:* $BD(189)

Mód: mind parancs, mind program módban használható.

Az e alapu exponenciális függvény kiszámítására szolgál (e=2.7182818..). x értéke megközelítőleg a (-88,88) intervallumba kell hogy essen.

Szintaxis: EXP (<aritmetikai kifejezés>)

Példák:

```
(i)    PRINT EXP(10):REM=22026.4658
(ii)   Y=EXP(1):REM Y=2.71828183.
(iii)  PRINT EXP(LOG(X)):REM=X
(iv)   FOR N=0 TO 10:P(N)=EXP(-M)/ FN FACT(N)
(v)    50 NT=NE*EXP(-B*EXP(-K*T)
```

Az SQR függvényhez hasonlóan az EXP függvény is a hatványozás speciális esete: EXP(Q)=2.7182818^Q. Sok esetben van szükség rá, ezért kényelmesebb, ha külön is használhatjuk.

Első két példánk az EXP közvetlen hatását mutatja. (ii) azt szemlélteti, hogy az EXP függvény a LOG függvény inverze, leszámítva természetesen a kerekítési hibákat. Az utolsó két képlet valószínűségi, illetve statisztikai programokban használható. Az első a Poisson-elosztást számítja ki, felhasználva az előzetesen már definiált FACT(N) függvényt. Az utolsó képlet egy ún. logisztikus növekedési függvényt definiál; a népesedési folyamatok modellezésében lehet felhasználni.

Magát az e számot sokféleképpen lehet definiálni. Egyik definíció sem elemi, lévén az e irracionális szám. e^x-et legegyszerűbben az 1 + x + x/2! + x/3! + ...hatványsorral lehet kiszámítani. Az e^x függvény deriváltja önmaga.

Hibalehetőség Ha az argumentum értéke túl nagy, akkor ?OWERFLOW ERROR hibajelzést kapunk. Ha az argumantum értéke túl kicsi, akkor hibajelzés nélkül a gépi nullával számol tovább az interpreter.

FOR..TO..[STEP]..

Röviditések: fO stE *Tokenek*: FOR $81(129) TO $A4(164) STEP $A9(169)

Mód: mind parancs, mind program módban használható.

Lehetővé teszi a FOR..TO..[STEP].. és a megfelelő NEXT utasítások közti programrész többszöri végrehajtását. A NEXT utasítás végrehajtásakor az interpreter ellenőrzi a változó értékét, megkeresi a hozzá tartozó FOR..TO..[STEP].. utasítást, a STEP részben definiált értékkel megnöveli a ciklusváltozó értékét, és ellenőrzi, hogy beleesik-e a TO-ban definiált értéktartományba. Ha igen, a vezérlés a FOR..TO..[STEP].. utasítást követő utasításra adódik; ha nem, a NEXT utáni elsőre.

Szintaxis:
FOR<változó>=<arit. kif.> TO <arit. kif.> [STEP<arit. kif.>]

A FOR utáni változót **ciklusváltozónak**, az egyenlőségjel utáni kifejezés értékét **kezdőértéknek**, a TO utáni kifejezés értékét **végértéknek**, a STEP utáni kifejezés értékét pedig **növekménynek** hívjuk. Ha a STEP hiányzik, az interpreter a növekményt 1-nek tekinti. A FOR és a NEXT közti programsorokat **ciklusmagnak** hívjuk.

Példák:

```
(i)   FOR J=1 TO 1000: PRINT ".";: NEXT
      FOR J=1 TO 1000: PRINT J;: NEXT
      FOR XJ=1 TO 1000: NEXT
```

Az (i) alatt felsorolt példák a legegyszerűbb ciklusokat mutatják be. A ciklusváltozó értékét a cikluson belül nem változtatjuk. Valahányszor a NEXT végrehajtódik J, illetve XJ értéke 1-gyel nő, ez lesz ugyanis (a STEP utasítás hiányában) a növekmény értéke. Amikor a ciklusból kilépünk, a J, illetve XJ értéke 1001.

```
(ii)  FOR I=0 TO 255:POKE 7680+I,I:POKE 38400+I,14:NEXTI
(iii) FOR I=255 TO 0 STEP -1:POKE 7680+I+1,PEEK(7680+I):NEXT I
```

Az (ii) parancs a képernyő memóriába írja be a számokat 0-tól 255-ig. Ennek hatására a képernyő tetején az összes lehetséges karakter kiíródik. A ciklusban használt változó mutatja, hogyan lehet biztosítani, hogy a ciklus ismétlődő végrehajtása során egyes mennyiségek értéke más és más legyen.

A (iii) parancssor a képernyő memória tartalmát tolja el egy karakterhellyel jobbra. Az egyes karakterek másolását hátulról előre kell végrehajtanunk, különben a képernyőn csak az első karakter maradna. Ezért kell használnunk a STEP -1 utasítást.

A ciklusok használata nem szokott különösebb problémát okozni, mégis néhány megjegyzést teszünk helyes használatukról. Először a szintaxisról. A ciklusváltozónak mindig **egyszerű valós** változónak kell lennie. FOR X%=1 TO 9 és FOR X(0)=0 TO 2 STEP -1 egyaránt helytelen. A ciklus a ciklusváltozó alsó és felső határaival is lefut. Például a FOR I=0 TO 5 ciklusutasítás az I=0,1,2,3,4,5 értékekkel hajtódik végre.

Ha a program a ciklusváltozó és a növekmény értékét pontosan tárolta, akkor a ciklus - néhány extrém esettől eltekintve - annyiszor fut, ahányszor matematikailag futnia kell. Általában az egész változók értékeit és a decimális törteket az interpreter pontosan tárolja. Ennek megfelelően a

```
FOR X=1 TO 1000 illetve a
FOR X=0 TO 10 STEP .0125
```

utasítások esetén a ciklusmag 1000-szer illetve 800-szor kerül végrehajtásra. A FOR X=1 TO 1000 STEP 1/3 utasítással már probléma lehet, célszerű a FOR X=1 TO 1000.1 STEP 1/3 utasítással helyettesíteni.

A BASIC interpreter megengedi a ciklusok egymásba ágyazását is. A következő program-séma illusztrálja ezt:

```
FOR X=X1 TO X2
 FOR Y=Y1 TO Y2
  FOR Z=Z1 TO Z2
    ...
   <ciklusmag>
    ...
  NEXT Z
 NEXT Y
NEXT X
```

A FOR utasítás végrehajtásakor 18 byte-nyi információ kerül a verembe. A FOR utasításon kívül a GOSUB is használja a vermet, együttes használatuknak így határt szab a verem nagysága (256 byte). Az interpreter minden egyes FOR utasítás végrehajtásakor ellenőrzi, hogy ez a ciklusváltozó szerepel-e a veremben. Ha igen, a 'verem teteje' mutató erre a változóra fog mutatni, a verem többi része elvész.

A ciklusok tetszőleges struktúrában egymásba ágyazhatók:

```
10 FOR X=XA TO XB: FOR Y=YA TO YB: NEXT Y
20 FOR A=AA TO AB: FOR C=CA TO CB: NEXT C,A,X
```

Az egyes programnyelvekben megtalálható *DO <utasítások> UNTIL <teszt>* konstrukció a következő ciklussal érhető el:

```
100 OK=-1:FOR J=KEZD TO 9E9
110 IF NOT OK THEN J=9E9:GOTO 200
125 <UTASITASOK>
150 IF <TESZT> THEN OK=0
175 <UTASITASOK>
200 NEXT J
```

A *DO <utasítások> WHILE <teszt>* konstrukció a következő ciklussal érhető el:

```
FOR J=-1 TO 0:<utasítások>:J=<teszt>:NEXT
```

Hibalehetőségek Az aritmetikai kifejezések kiértékelése számos hibát eredményezhet. A ciklusutasítás használatában már tervezési hibák is előfordulhatnak. Ezek közül a leggyakoribbak:

(i) A negativ növekmény lemarad.
(ii) NEXT hiányzik.
(iii) Az egymásba ágyazott ciklusváltozók felcserélődnek.
(iv) Ugyanazt a változót két egymásba ágyazott ciklusban használjuk.
(v) RETURN nélküli GOSUB könnyen okozhat ?NEXT WITHOUT FOR hibaüzenetet.
(vi) Ugyanezt a hibaüzenetet kapjuk abban az esetben, ha a NEXT utasításban nem létező ciklusváltozót használunk.
(vii) Ha egész, sztring vagy tömbváltozót használunk ciklusváltozónak ?SYNTAX ERROR hibaüzenetet kapunk.

FRE

Rövidítés: fR *Token*: $A7(167)

Mód: mind parancs, mind program módban használható.

Kiszámítja, hány szabad byte található a BASIC munkaterületen, azaz mennyi a különbség a tömbök vége és a sztringek eleje közt. Ezt megelőzően azonban egy ún. szemétgyűjtést végez.

Szintaxis: FRE(<kifejezés>)

FRE alakját tekintve függvény, de az argumentum kiszámítására vagy ellenőrzésére nem kerül sor. FRE valójában egy nullaváltozós függvény. Általában PRINT FRE(0) vagy F=FRE(0) alakban használjuk. Az FRE(X$), FRE(A+X%), FRE(X) kifejezések szintaktikusan helyesek.

Példák:

```
(i)  PRINT "SZABAD BYTE-OK SZAMA=";FRE(0)
(ii) X=FRE(0):DIM X%(278):PRINT FRE(0)-X
```

Az első példa egyszerűen kiírja a szabad memóriakapacitást. A második esetben a FRE függvényt annak kiszámítására használjuk, hogy egy sztring tömb definiálása a memóriában hány byte-ot foglalt le.

A BASIC munkaterületen a program egyes részei a következőképpen helyezkednek el:

BASIC program egyszerű változók tömbök szabad memória sztringek

eleje BASIC munkaterület vége

Az alábbi példa egy beolvasó rutin, amely 20 egymás utáni karaktert olvas be a billentyűzetről:

```
10 FOR I=1 TO 20
20 GET X$: IF X$="" GOTO 20
30 I$=I$+X$: NEXT
40 X=FRE(0)
```

X$ minden esetben 1 hosszúságú és I$ minden egyes kiszámítása I$ számára a sztringek közt újabb helyet foglal le. A 30. sor I.-ik végrehajtásakor a memória I*(I+3)/2 byte-ját foglalják le X$ és I$ előző értékei. A 20 karakter beolvasása a memóriában összesen 230 byte-ot foglal el. A 40. sorban az FRE(0) kiszámításakor X$ és I$ felesleges darabjai 'eltűnnek'.

Hibalehetőség: Nincs.

GET és GET#

Rövidítés: gE és gE# *Token:* $A1(161)

Mód: csak program módban használható.

Az utasításban megadott input perifériáról egyetlen byte-ot olvas be. Billentyűzet esetén, ha egyetlen karakter sincs a billentyűzet-pufferben, az eljárás az üres sztringgel tér vissza.

Szintaxis: GET [# <aritmetikai kifejezés>] <változólista>

A <változólista> legalább egy elemet kell hogy tartalmazzon; elemei általában sztring változók. Az aritmetikai kifejezés az input file logikai file számát definiálja. A GET és a # jel közt nem lehet szóköz.

Példák:

```
(i)  5 GET X$:IF X$="" THEN GOTO 5
     10 PRINT" ";X$;" ";ASC(X$):GOTO 5
```

```
(ii) 200 GET A$,B$,C$:PRINT A$+B$+C$:GOTO 200
```

Az első példánk segítségével kipróbálhatjuk, hogy az egyes billentyűket a GET hogyan is olvassa be. Az 5-ös sorszámú utasításban addig várunk, míg legalább egy karakter nem kerül a billentyűzet-pufferbe (**dinamikus megállás**), azután ezt 'visszaírjuk' a képernyőre, és kiírjuk az ASCII kódját is. érdemes kipróbálni a <RETURN>, <DEL>, <CTRL-RVSON> stb. billentyűket!

A (ii) példában három byte-ot olvasunk be egy végtelen ciklusban.

A GET utasítás - szemben az INPUT utasítással - byte-onként olvassa be az adatokat, ezért alkalmas ismeretlen struktúrájú rekordok olvasására is. Lemezes file-ok esetén ezt legegyszerűbben az alábbi ciklussal tehetjük meg:

```
(iii) 1000 GET#8,X$:IF ASC(X$)=13 GOTO 3000
      1010 IN$=IN$+X$:REM SZTRING OLVASASA
      1020 GOTO 1000
```

Megjegyezzük, hogy az 50 GET A utasítás szintaktikusan helyes. Ha azonban a byte értéke nem a 0-9 intervallumba esik, akkor egy ?SYNTAX ERROR hibaüzenetet kapunk. Ha a következő byte ; : vagy , akkor ?EXTRA IGNORED hibaüzenetet kapunk, és A értéke 0 lesz. Célszerübb a GET A$ utasítás alkalmazása, majd azt követően annak ellenőrzése, hogy számjegyet olvastunk-e be.

Billentyűzet-puffer A GET utasítás az inputpuffer első karakterét olvassa be. Ezeket a karaktereket a hardver megszakító rutin helyezi el a pufferba. (Másodpercenként kb. 50-szer hajtódik végre.) A billentyűzet-puffer a 1319-1328 címeken található. A 239 címet használja az interpreter a pufferben levő karakterek (byte-ok) számának tárolására. A puffert 'kiüríthetjük' a POKE 239,0 utasítással, vagy a

```
10 GET X$ : IF X$>"" THEN 10
```

utasítással. A billentyűzet-puffer léte a következő program végrehajtásával érzékeltethető:

```
10 FOR J=0 TO 8000:NEXT:
   FOR J=0 TO 20:GET X$:PRINT X$:NEXT
```

A RUN parancs kiadása után - amig az első sor fut - billentyűzzünk be néhány karaktert. A képernyőn csak az elsőként beírt tízet látjuk viszont.

Szalagos file Abban az esetben, ha a GET# utasításban definiált logikai file szám egy kazettás file-ra utal, a karaktert a kazetta-pufferből olvassa a GET# rutin, nem pedig a billentyűzet-pufferből. Ha a kazetta-puffert már teljes egészében beolvastuk, a program futását az interpreter felfüggeszti, és a következő rekordot betölti a szalagról. A 'szalag vége' jelnek egy nulla byte felel meg. Ennek olvasása az ST állapotjelző byte-ot 64-re állitja. Ha ST értékét nem ellenőrizzük, akkor a következő GET# (vagy bármilyen, a szalagra vonatkozó input utasítás I/O utasítás) ST értékét nullázza és a kazettáról további adatok olvashatók.

Hibalehetőségek A GET# utasításban szereplő logikai file-t előbb meg kell nyitnunk. Problémát szokott okozni, ha az ST vizsgálata elmarad, és így esetleg az EOF jel után is tovább olvassuk a file-t. Ekkor a lemezegység teljes egészében 'lemerevedhet'.

GETKEY

Rövidítés: getkE *Token*: nincs

Mód: csak parancs módban használható.

A program futása felfüggesztődik egészen addig, míg nem nyomunk le egy billentyűt. Ezután ennek ASCII kódja, mint egy egyhosszúságú sztring a GETKEY-t követő változóba kerül.

Szintaxis: GETKEY <változólista>

A <változólista> egymástól vesszővel elválasztott változókat tartalmazhat.

Példák:

```
(ia)   10 GETKEY A$
(ib)   10 GET A$: IF A$="" THEN GOTO 10

(ii)   1000 PRINT "SZAM1=?";: X$=""
       1010 GETKEY A$
       1020 IF A$=CHR$(13) THEN X=VAL(X$): RETURN
       1030 IF A$<"0" OR A$>"9" THEN GOTO 1010
       1040 X$=X$+A$
       1050 ?CHR$(157)+A$+"?";: GOTO 1010

(iii)  260 GETKEY A$: GRAPHIC 0
```

Az (ia), illetve az (ib) példák a GET és a GETKEY viszonyát mutatják be. A GETKEY - úgy ahogy az (ib) alatti IF utasítás mutatja - egészen addig vár, míg legalább egy billentyűt le nem nyomunk.

A GETKEY lehetőséget nyújt ellenőrzött inputra. A (ii) alatt látható alprogram segítségével egész számokat olvashatunk be. Számjegyen kívül a rutin egyéb karaktert nem fogad el. A beírást a <RETURN> leütése fejezi be. A kurzor helyett egy ? látható.

A (iii) példa a GETKEY egy másik tipikus használatát mutatja. A nagyfelbontású képernyőre a program eleje rajzolt valamit. A kép egészen addig látható, míg meg nem nyomunk egy billentyűt. Ekkor a 260-as programsor visszaváltja a karakteres képernyőt.

Hibalehetőségek A GETKEY A utasítás szintaktikusan helyes. Ha azonban nem számjegy billentyűt nyomunk meg, akkor ?SYNTAX ERROR hibaüzenetet kapunk.

GO

Rövidítés: nincs *Token:* $CB(203)

Mód: mind parancs, mind program módban használható.

A GO utasítás lehetővé teszi a GOTO utasítás GO TO alakban való írását.

Szintaxis: GO <szóközök> TO

Hibalehetőségek Ugyanazok, mint a GOTO utasításnál. A GOTO utasítás GO TO alakban való írása lassítja a program futását, ezért inkább az egybeírt alak használatát javasoljuk.

GOSUB

Rövidítés: goS *Token:* $8D(141)

Mód: mind parancs, mind program módban használható.

Végrehajtja a GOSUB után megadott sorszámú programsorban kezdődő alprogramot. Ez azt jelenti, hogy feltétel nélküli vezérlésátadás jön létre a GOSUB utasításban megadott számú sorra. Amikor ezt követően a legközelebbi RETURN utasításhoz ér az interpreter, a vezérlés visszakerül a GOSUB utasítást követő első utasításra.

Szintaxis: GOSUB <sorszám>

A <sorszám> csak előjel nélküli egész konstans lehet.

Példák:

(i)
```
FOR V=0 TO 21: FOR H=0 TO 22: GOSUB 600: NEXT H,V
600 <utasítasok>
```

(ii)
```
100 E=0: FOR I=1 TO LEN(S$)
110 IF MID$(S$,I,1)=":" THEN E=E+1
115 NEXT
120 GOSUB 6000
125 <utasítások>
...
6000 IF E=0 THEN A$=UZEN$(0)
6010 IF E>0 THEN A$=UZEN$(1)
6020 PRINT" ";A$: FOR I=1 TO 2000: NEXT
6030 RETURN
```

```
(iii) 5000 GOSUB 5010
      5010 REM *** CSIPOGO ***
      5020 <csipogo program utasitasai>
```

Első példánk azt mutatja, hogy hogyan használható a GOSUB utasítás parancs módban. A 600 sorszámmal kezdődő (a példában konkrétan nem szereplő) szubrutin a kurzor helyét változtatja a képernyőn. A parancs minden lehetséges értékre ellenőrzi a rutin helyességét.

A (ii) példa egy ellenőrző rutin része. Megvizsgálja, hogy az inputként beolvasott S$ sztring tartalmaz-e kettőspontot (:). Ezután kerül sor a 6000 alatt kezdődő alprogram végrehajtására, ami a vizsgálat eredményétől függően beállítja A$ tartalmát, és kiírja a képernyőre. A program ezek után folytatódik.

Az utolsó példa egy olyan szubrutint mutat be, aminek két belépési pontja van: 5000, illetve 5010. Ha a főprogramból az alprogramot GOSUB 5000-rel hívjuk meg, az 5020 alatti programrész – az 5000 alatti újbóli GOSUB miatt – kétszer hajtódik végre, a program kétszer 'csipog'. Ha a GOSUB 5010 utasítást használjuk, a program csak egyszer 'csipog'.

Még abban az esetben is, ha egy programrészt csak egy helyen akarunk a programban használni, előfordulhat, hogy célszerű alprogramként kidolgozni. Ilyenek például a következő programrész alprogramjai:

```
7000 IF S$="S" THEN GOSUB 5000: GOTO 6000
7010 IF S$="L" THEN GOSUB 5100: GOTO 6000
7020 IF S$="@" THEN GOSUB 5200: GOTO 6000
7030 IF S$="$" THEN GOSUB 5300: GOTO 6000
7040 GOTO 6000
```

Általában, ha egy programrész hosszabb, mint egy sor, vagy többszörösen egymásba ágyazott IF utasításokat tartalmaz, célszerű megfelelően kialakított alprogramokat használni.

A GOSUB utasítások **egymásba ágyazhatók.** Ennek azonban határt szab nemcsak a program áttekinthetősége, hanem a verem nagysága is. Ilyen esetekben ügyelnünk kell arra, hogy a RETURN utasítás mindig csak a *legutolsó* GOSUB után tér vissza, s ezért a vezérlést pontosan meg kell terveznünk. Példa erre a következő program, mely elsősorban az ilyen programok *struktúráját* mutatja be.

```
   10 GOSUB 5000
   20 <SZAMITASOK>
      ...
 5000 <SZAMITASOK>
      GOSUB 10000
      IF E=0 THEN RETURN
      ...
      <SZAMITASOK>
      E=3: RETURN
10000 <SZAMITASOK>
      IF <TESZT> THEN E=0: RETURN
      <SZAMITASOK>
```

Az 5000. sorban kezdődő alprogram egy további alprogramot (10000) hív meg. Ez az utóbbi hiba esetén az E értékét 0-ra állítja. Ebben az esetben a vezérlés azonnal visszatér a 10.sor alá. Mivel az 5000 alatti alprogram sikeres visszatérés esetén E értékét maga is beállítja, ezt a 20-as sorral kezdődő programrészben is felhasználhatjuk.

Hibalehetőségek Ha a GOSUB utasításban megadott programsor nem létezik, akkor ?UNDEF'D STATEMENT ERROR hibajelzést kapunk. A sorszám előállitása az első nem numerikus karakterig tart. Igy például GOSUB 12I0 hibajelzés nélkül végrehajtódik, és ekvivalens a GOSUB 12 utasítással. Amennyiben a veremben már nincs elég hely, ?OUT OF MEMORY ERROR hibaüzenetet kapunk.

100 GOSUB 100 például mindig ezt a hibajelzést adja, szemben a 100 GOTO 100 programrésszel, ami hibajelzés nélküli végtelen ciklust eredményez.

GOTO

Rövidítés: gO *Token*$89(137)

Mód: mind parancs, mind program módban használható.

Feltétel nélküli vezérlés átadás. A program a GOTO utasításban megadott számú sor végrehajtásával folytatódik.

Szintaxis: GOTO <sorszám>

Példák:

```
(i)   D$="1234": GOTO 1500
(ii)  100 GET X$: IF X$="" GOTO 100
      110 IF X$=CHR$(13) GOTO 300
      120 <utasítások>
(iii) GO TO 2000
```

Az első példa a GOTO direkt (parancs) módban való használatát mutatja. A D$ beállitása után a program futása az 1500-as sorszámu programsorral folytatódik. Az utasitás hatása nem azonos RUN 1500-zal, mert az előbb töröl majdnem mindent, például a D$ értékét is.

A második példa egy rövid ciklust tartalmaz. Amint megnyomunk egy billentyűt, az IF feltétel hamissá válik, és a GOTO 100 utasitás helyett a következő sorra(120) kerül a vezérlés.

A (iii) példa egyszerűen illusztrálja a GO TO alak használatának lehetőségét.

Hibalehetőségek A <sorszám> előállítása az első nem numerikus karakterig tart. Az így előállított sorszámra kisérli meg átadni a vezérlést az interpreter. Például GOTO 1X ekvivalens a GOTO 1 utasítással. Nem létező sorszám esetén ?UNDEF'D STATEMENT hibajelzést kapunk.

Rosszul tervezett programok esetén könnyen előfordulhat, hogy nem megfelelő a vezérlés. Ilyen esetben a program struktúráját újból kell ellenőrizni (majd a szükséges változtatásokat végrehajtani).

GRAPHIC

Rövidítés: gR *Token*:$DE(222)

Mód: mind parancs, mind program módban használható.

Segítségével beállíthatjuk a képernyő kijelzési módját. Első paraméterének értéke megadja, hogy a kijelzés:

a/ karakteres, nagyfelbontású vagy vágott legyen-e;
b/ normál vagy többszínű üzemmódot használjunk-e.

A második paraméter értéke megadja, hogy a parancs kiadásával egyidőben a megfelelő képernyők törlődjenek-e vagy sem.

Szintaxis: GRAPHIC <arit.kif.1> [, <arit.kif.2>]

Az első aritmetikai kifejezés értéke a következő kijelzési módoknak felel meg:

érték	**Kijelzési mód**
0	Normál, karakteres üzemmód
1	Normál, nagyfelbontású üzemmód
2	Normál, vágott képernyő
3	Többszínű, nagyfelbontású üzemmód
4	Többszínű, vágott képernyő

A második aritmetikai kifejezés értéke 0 vagy 1 lehet. Ha 1, az érintett képernyők törlődnek. Ha 0, akkor az előző tartalmuknak megfelelő információ jelenik meg.

Példák:

```
(i)     GRAPHIC 1,1
(ii)    GRAPHIC 2,1
(iii)   GRAPHIC 3
```

Úgy hisszük a példák önmagukért beszélnek. Az első két esetben a képernyők törlődnek is, míg a harmadik esetben nem. A (ii) példában **vágott képernyőt** használunk, ami azt jelenti, hogy a **karakteres képernyő utolsó öt sora látszik, míg a nagyfelbontású képernyő első 160 rasztersora(=20 karaktersor).**

Hibalehetőségek Az aritmetikai kifejezés kiértékelése közben számos hiba történhet. Ha értékük nem esik a jelzett intervallumba, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

GSHAPE

Rövidítés: gS *Token:*$E3(227)

Mód: mind parancs, mind program módban használható.

Az SSHAPE utasítással definiált - téglalap alakú - képet másolhatjuk vissza a nagyfelbontású képernyő tetszőleges helyére.

Szintaxis:
GSHAPE <sztringváltozó> [, [<x>,<y>] [,<mód>]]

A téglalap jobb felső sarkának koordinátáit az <x>,<y> koordinátapár határozza meg. Ha elhagyjuk, akkor a grafikus kurzor pillanatnyi helyével számol az interpreter. A <mód> paraméter a képernyőre való kiírás módját határozza meg. Ha a képernyő egy adott (raszter)pontjának az értéke R, s a GSHAPE utasításban szereplő sztring ennek a pontnak megfelelő bitjének értéke Y, akkor a <mód>-tól függően R új értéke a következő lesz:

<mód>	**a pont(R) új értéke**	
0	Y	(másolás)
1	1-Y	(inverz alak)
2	Y OR R	(logikai vagy)
3	Y AND R	(logikai és)
4	Y EOR R	(logikai kizáró vagy)

Ha a <mód> értékét nem adjuk meg, akkor az interpreter 0-val számol tovább.

Példák:
```
10 GRAPHIC 1,1
20 CHAR ,1,1,"A"
30 DRAW ,12,6 TO 14,5
40 SSHAPE A$,8,5,16,16
50 CHAR ,1,1,"O"
60 SSHAPE O$,8,5,16,16
70 CHAR ,1,1,"U"
80 SSHAPE U$,8,5,16,16
90 GRAPHIC 1,1
95 FOR X=3 TO 21 STEP 2
100 GSHAPE U$,80,8*X-3: CHAR ,11,X,"RY L"
110 GSHAPE A$,120,8*X-3: CHAR ,16,X,"SZL"
115 GSHAPE O$,152,8*X-3
116 NEXT X
120 GETKEY A$: GRAPHIC 0
```

A fenti program futásának eredménye:

ÚRY LÁSZLÓ
ÚRY LÁSZLÓ
ÚRY LÁSZLÓ
ÚRY LÁSZLÓ
ÚRY LÁSZLÓ
ÚRY LÁSZLÓ
ÚRY LÁSZLÓ
ÚRY LÁSZLÓ
ÚRY LÁSZLÓ
ÚRY LÁSZLÓ

Hibalehetőségek A sztring- és aritmetikai kifejezések kiértékelése közben számos hiba adódhat. Ha a <mód> értéke nem esik a 0-4 intervallumba, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

HEADER

Rövidítés: heA *Token*:$F1(241)

Mód: mind parancs, mind program módban használható.

Lehetőséget nyújt lemezek megformázására és törlésére.

Szintaxis

HEADER <sztring.kif.>, D<arit.kif.1> [,I<id.>] [{, | ON} U<arit.kif.2>]

A <sztring.kif.> a lemez új neve lesz. Ha nem sztring konstans, akkor kerek zárójelek közé kell tenni. A két aritmetikai kifejezés a meghajtó, illetve a lemezegység hardver egységszáma. Az I paraméterben szereplő <id.> két karakteres azonosító. Ha elmarad, az utasítás nem formázza meg, csak törli a lemezt. Újonnan vásárolt lemezeket - kivéve természetesen ha program van rajtuk - az I paramétert is használó HEADER utasítással kell megformázni.

Példák:

```
(ia)     HEADER "PROGRAMOK",DO
(ib)     A$="PROGRAMOK"
         HEADER (A$),DO
(ii)     HEADER "PROGRAMOK",DO,IAA
(iii)    HEADER "ADATOK.2",D1 ON U9
```

Az (ia) példa a 8-as lemezegység 0-ás meghajtójában levő - már megformázott - lemez tartalmát törli, és a lemezt "PROGRAMOK" névre nevezi át. Az (ib)-ben alatt látható két parancs ugyanezt a feladatot oldja meg, de a HEADER parancsban most sztringet használunk. Ennek megfelelően kerek zárójelek közé kell tenni.

A (ii) példa a (8-as lemezegység) 0-ás meghajtójában levő lemezt formázza meg. Ezt követően a lemez már használható programok, illetve adatok tárolására.

(iii) az egységszám megadására mutat példát.

Tekintettel arra, hogy a HEADER parancs a lemezen esetleg rajta levő információt teljes egészében megsemmisíti, a HEADER utasítás kiadása után az interpreter visszakérdez:

ARE YOU SURE?

Ha erre az <Y> megnyomásával válaszolunk, az utasítást az interpreter végrehajtja. Ha az <N> megnyomásával válaszolunk, az utasítás figyelmen kívül marad.

Hibalehetőségek Ha a <sztring> nem sztring konstans, és nem tettük kerek zárójelbe, akkor ?SYNTAX ERROR hibajelzést kapunk. Ha olyan lemezegységre hivatkozunk, ami nincs a rendszerben, akkor ?DEVICE NOT PRESENT ERROR hibajelzést kapunk. Vigyázzunk arra, hogy csak olyan lemezt töröljünk le, amelyikre nincs szükségünk! Ha a lemezen nincs kivágva az írást engedélyező rés, akkor nem kapunk hibajelzést, csak a lemezegység lámpája jelzi a hibát. A törlés és a formázás természetesen nem hajtódik végre.

HEX$

Rövidítés: hE *Token*:$D2(210)

Mód: mind parancs, mind program módban használható.

Sztring függvény, amelyik az argumentumaként megadott szám hexadecimális alakjával tér vissza.

Szintaxis: HEX$(<arit.kif.>)

Az aritmetikai kifejezésnek a 0<=N<=65535 intervallumba kell esnie.

Példák:

```
(i)    PRINT HEX$(32) : REM = "20"
       PRINT HEX$(45) : REM = "2C"
(ii)   ? HEX$(DEC("FFFF")) : REM = "FFFF"
```

Az (i) alatti példák HEX$ közvetlen hatását mutatják. (ii) azt példázza, hogy HEX$ a DEC inverze. Ennek megfelelően bármilyen hexadecimális szám is áll a "FFFF" helyén, a (ii) példa mindig azt nyomtatja.

Hibalehetőségek Az aritmetikai kifejezés kiértékelése közben számos hiba adódhat. Ha értéke nem esik a jelzett intervallumba, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

IF - - - THEN

Rövidítés: nincs *Token*:$8BC(139)

Mód: mind parancs, mind program módban használható.

Feltételes vezérlésátadás. Ha az IF kulcsszót követő feltétel teljesül, akkor a feltételt követő utasítás kerül végrehajtásra; ha nem, akkor a következő programsor.

Szintaxis:

```
                                          ┌      ┌ <sorszám>  ┐ ┐
                                          │ THEN ┤            ├ │
   IF <aritmetikai kifejezés>            ┤      └ <utasítás> ┘ ├
                                          │                     │
                                          └ GOTO  <sorszám>     ┘
```

<sorszám> csak előjel nélküli konstans lehet!

Példák:

```
(i)    FOR I=1 TO 1000: X=RND(1): GOSUB 1000:IF T="*" THEN NEXT
(ii)   100 IF P=72 THEN P=0: GOSUB 1200
(iii)  200 IF X=1 THEN IF A=0 AND B=0 THEN GOSUB 300
(iv)   600 IF (X<0) AND (X>0) THEN IDE NEM KERUL A VEZERLES!!!
```

Az első példánk az 1000-rel kezdődő alprogramot ellenőrzi. Az X=RND(1) utasítás X értékét véletlenszerűen állítja be. Az alprogram használja az X értékét és visszatéréskor beállítja a T$ értékét. A ciklus abban az esetben jár le teljes egészében, ha T$="*" minden szubrutinhívás esetén teljesül. Példánk mutatja, hogyan lehet az IF utasítást parancs módban is használni.

A (ii) példa ellenőrzi, hogy P értéke elér-e egy bizonyos határt (72), s ha igen, akkor ezt 0-ra változtatja, és meghívja az 1200 alatt kezdődő alprogramot.

Következő példánk azt mutatja, hogy az IF utasítások egymásba ágyazhatók.

Az utolsó programsor egy hibás program része, a feltétel sohasem lesz igaz, ezért a vezérlés a THEN utáni - szintaktikusan helytelen - részre sohasem kerül.

Hibalehetőségek A kifejezés kiértékelése közben számos hibajelzést kaphatunk. A GO TO alak nem megengedett; IF X<>0 GO TO 10 tehát szintaktikus hibát eredményez. Ha a GOTO egy nem létező sorra mutat, akkor ?UNDEF'D STATEMENT ERROR hibajelzést

kapunk. Ha aritmetikai kifejezést használunk a feltétel helyén, akkor csak a 0 számít hamisnak. IF X<>0 THEN... és IF X THEN... tehát ekvivalensek.

INPUT

Rövidítés: nincs *Token*:$85(133)

Mód: csak program módban használható.

Az utasítás lehetővé teszi, hogy a billentyűzetről közvetlenül **adatokat adjunk át egy futó BASIC programnak.** Az INPUT utasítás a billentyűzetről bevitt értékeket mindig **visszaírja** a képernyőre. Az eljárás a <RETURN> megnyomásával fejeződik be.

Szintaxis: INPUT ["<szöveg>";]<változólista>

A <szöveg> tetszőleges karaktersorozat lehet; a <változólista> elemei egymástól vesszővel elválasztott egyszerű vagy tömbváltozók. Az opcionális <szöveg> karaktersorozat, majd azt követően egy kérdőjel jelenik meg a képernyőn, végül ezt követi a villogó kurzor. A bebillentyűzött értékek feldolgozása a következőképpen történik:

1/ Az interpreter a legtöbb karaktert az input változók részének tekinti, bizonyos jeleket azonban ettől eltérően értelmez. Ilyenek elsősorban az " , : jelek. A " jel utasítja az interpretert, hogy az azt követő jeleket egy sztring részének tekintse. A <RETURN> utasítás mindig befejezi az input sor bevitelét.

2/ Az INPUT utasítást végrehajtó rutinnak közös részei vannak a GET és a DATA utasítások hasonló részeivel, így a vesszőt (,) és a kettőspontot (:) **elválasztó jel**ként értelmezi.

3/ Az INPUT utasítás az input sort a <szöveg> és a kérdőjel (?) kinyomtatásától a logikai sor végéig (maximum 88 karakter) dolgozza fel.

Példák:

(i)
```
100 INPUT "HONAP";H$
200 INPUT "CIM /VESSZOT NEM HASZNALHAT!/";C$
300 INPUT X,Y:REM KEZDOPONT
```

Fenti példák az INPUT utasítás legegyszerűbb használatát mutatják; használata ilyenkor nem okozhat problémát. *Helytelen adatbevitel* esetén azonban furcsa dolgok történhetnek. A <HOME> lenyomása a képernyő bal felső sarkába állítja a kurzort. A

<RETURN> azonnali terminálást eredményez, a <RUN> elkezd betölteni a lemezről egy új programot stb.

```
(ii)  1000 INPUT AA$ BB,C%:REM KEZDOERTEKEK
(iii) 2000 FOR J=0 TO 10:INPUT X$(J):NEXT
      2010 FOR J=0 TO 10:PRINT X$(J):NEXT
```

Az 1000 sorszámú utasítás három input változót tartalmaz. A

KUTYA, 56.83,7.1 <RETURN>

helyes válasz. A program futása az AA$="KUTYA", BB=56.83, C%=7 értékekkel folytatódik. (Az egész változóra vonatkozó értékadás a szokásos kerekítési szabályok segítségével történik.) A következő válasz egy ?EXTRA IGNORED hibaüzenetet eredményez:

KUTYA,18,56.83,7.1 <RETURN>
A

KUTYA <RETURN>

válasz a következő sor elején két kérdőjel (??) és a kurzor kinyomását eredményezi; jelezve, hogy további értékeket kell bevinnünk a gépbe.

Következő példáink azt mutatják, hogyan lehet az INPUT utasítást néhány egyszerű trükkel biztonságosabbá és kényelmesebbé tenni.

```
(iv)  10 INPUT "<CLR>N=";N: PRINT N
      20 INPUT "NEV=   <USPC><B><B><B>";X$:PRINT X$
(v)   30 INPUT"ALFA  2<B><B><B>";A:PRINT A
(vi)  200 POKE 198,3:POKE 631,34:POKE 632,34:POKE 633,20
      210 INPUT X$:PRINT X$
```

Az első, (iv) alatti példában az input promptban elhelyezett vezérlő karakter hatására a képernyő törlődik, s utána kéri a program N értékét.

A 20. sorban levő példában az input sztringbe a következő vezérlő karaktereket helyeztük el: <B>=<CRSR BALRA>, <USPC>=<SHIFT-szóköz>. A <RETURN> azonnali megnyomásának a hatására az X$-ba nem az előző eredmény, hanem egy egyetlen karaktert tartalmazó sztring kerül.

A (v) példában a három <CRSR BALRA> hatására az INPUT utasítás kurzora éppen a 2 felett fog villogni. Ha ez az érték megfelel, akkor csak a <RETURN>-t kell megnyomni, magát az értéket nem kell beírni.

A 200-210-es sorokban található rutin a billentyűzet-pufferbe

három karaktert (""<DEL>) helyez el. A ? kiírása után a három karaktert az interpreter beolvassa. Ez azt eredményezi, hogy az inputsor már nem lehet üres, és a második " utasítja az interpretert, hogy sztringnek tekintse az inputot. Igy az vesszőt (,) és kettőspontot (:) is tartalmazhat. A két utasítás együtt valójában egy INPUTLINE X$ utasítást szimulál.

Hibalehetőségek Ez a meglehetősen hosszú rutin párhuzamosan dolgozza fel az input sort és az INPUT utasításban levő változólistát. Az inputsorból előállított értékeket megkísérli hozzárendelni a soron következő input változóhoz. Ha a típus nem egyezik, akkor ?REDO FROM START üzenettel az interpreter újra végrehajtja a **teljes** INPUT utasítást. Ha az inputsor kevesebb értéket tartalmaz, mint ahány INPUT változó van, akkor a következő sor elejére két kérdőjel (??) íródik ki, és a rendszer a **hiányzó** értékek bevitelét várja. Ha az inputsor több értéket tartalmaz, mint ahány változó van az INPUT utasításban, akkor ?EXTRA IGNORED üzenetet kapunk, és a program futása folytatódik.

Abban az esetben, ha a CMD utasítás segítségével megváltoztattuk az elsődleges output file-t, a <szöveg> a CMD-ben definiált file-ba íródik ki, a kurzor azonban a *képernyőn* jelenik meg.

Az utasítás azért nem használható direkt módban, mert a végrehajtás igénybe veszi az input-puffert. Ha mégis megkíséreljük, akkor ?ILLEGAL DIRECT ERROR hibaüzenetet kapunk.

INPUT#

Rövidítés: iN (a # jel már nem kell!) *Token:* $84(132)

Mód: csak program módban használható.

Lehetővé teszi a háttértárakon rögzített adatok visszaolvasását.

Szintaxis: INPUT# <aritmetikai kifejezés>, <változólista>

Az aritmetikai kifejezés értékének az 1-255 intervallumba kell esnie. Ez annak a file-nak a **logikai file száma**, amelyikből az adatokat be kívánjuk olvasni. A <változólista> egymástól vesszővel elválasztott egyszerű vagy tömbváltozókat tartalmazhat. Az INPUT és a # között nem lehet szóköz.

A file-ból beolvasott karaktereket az interpreter a következő szabály szerint dolgozza fel:

Az alfanumerikus jelek a beolvasott értékek részét képezik,

hasonlóan, mint az INPUT utasítás esetében. A <RETURN> (azaz CHR$(13)) olvasásának ugyanaz a hatása, mint a <RETURN> billentyű benyomásáé az INPUT utasítás esetén. Hasonlóan a vessző (,) és a kettőspont (:) az egyes tételek végét jelzik (kivéve idézőjel után). Az ?EXTRA IGNORED hibajelzéshez hasonló üzenetet nem kapunk, és általában egyetlen adat sem vész el. Az input-puffer határt szab az egyetlen INPUT# utasítással beolvasható értékeknek; ezek maximum 87 adatbyte-ból és a CHR$(13) jelből állhatnak.

Példák:

```
(i)   10 OPEN 4,1,1
      20 FOR J=1 TO 10:INPUT X$:PRINT#4,X$:NEXT
      30 CLOSE 4
      40 OPEN 5,1,0
      50 FOR J=1 TO 10:INPUT#5,Y$:PRINT Y$:NEXT
      60 CLOSE 5
```

A fenti példa egy - a kazettás egységre vonatkozó - ki/beviteli utasításpár, amelyen nem tüntettük fel a szalag visszacsévélési, az ST ellenőrző részeket. A file számok (ami az írás esetén 4, az olvasás esetén 5) természetesen tetszőlegesek lehetnek, de hogy jobban látszódjék, hogy különböző célra használjuk őket, nem azonosak az egységszámmal. Az (i) alatti program a billentyűzetről 10 számot olvas be, majd ezeket egy név nélküli szalagos file-ba írja ki. A (ii) alatti programrész ezeket a számokat visszaolvassa az INPUT#5 utasítás segítségével, és kiírja a képernyőre.

```
(ii)   10 OPEN 1,0: REM FILE#1=A BILLENTYUZET
       20 OPEN 3,3: REM FILE#3=A KEPERNYO
       30 INPUT#1,X$: PRINT#3,X$: GOTO 30
```

A (ii) alatti példák azt mutatják, hogyan kezelhetjük a billentyűzetet file-ként. Ebben az esetben nem az INPUT, hanem az INPUT# utasítást kell az adatok beolvasására használni.

```
(iii) 5 OPEN 1,8,2,"@:PROBA,S,W"
       10 FOR J=1 TO 10: PRINT#1,STR$(J): NEXT
       20 CLOSE1
       30 OPEN 2,8,3,"PROBA,S,R"
       40 FOR J=1 TO 10: INPUT#2,X$: PRINT X$: NEXT
       50 CLOSE2
```

Az utolsó (iii) programrész egy lemezes file-ba való írást, majd ugyanennek a file-nak a visszaolvasását mutatja be.

Az INPUT és INPUT# utasítások a billentyűzetről, illetve a megfelelő file-ból kapott karaktereket az input-pufferbe ($0200)

másolják. A pufferba való másolást a <RETURN> billentyű lenyomása, illetve egy CHR$(13) olvasása fejezi be. A <RETURN> byte helyett azonban egy CHR$(0) kerül az input-pufferbe. A $01FF címen egy vessző található, ez lehetővé teszi, hogy az inputsor első tételét az interpreter ugyanúgy dolgozza fel, mint a többiakat. Az INPUT-ról szóló rész egyik példáját az interpreter a pufferben így helyezi el:

```
$1FF $200 1 2 3 4 5 6 7 8 9 A B C D E F 0 1 2
  ,    K  U T Y A , 1 8 , 5 6 . 8 3 , 7 . 1 null
```

A tételek feldolgozása hasonló mint az INPUT és a READ utasítások esetén.

Hibalehetőségek A szintaktikus hibákon kívül leggyakoribb, hogy a file végén túl akarunk olvasni, vagy a rekordokat nem megfelelően tagolva olvassuk vissza. Ha az input változó típusa nem felel meg a file-ból beolvasott adatnak, akkor ?BAD DATA hibajelzést kapunk. (Az INPUT utasítás ilyenkor küldi a REDO FROM START üzenetet. File-ok esetében ennek nincs értelme.)

INSTR

Rövidítés: inS *Token*:$D4(212)

Mód: mind parancs, mind program módban használható.

Kétváltozós függvény, amelyik megadja, hogy az első paraméterként szereplő sztringben hol kezdődik a második sztring. Ha egyáltalán nem szerepel, akkor a függvény értéke 0 lesz. Az összehasonlítás a harmadik paraméterként megadott pozíciótól kezdődik.

Szintaxis:
INSTR (<sztring.kif.1>, <sztring.kif.2> [,<arit.kif.>])

Példák:

```
(i)     PRINT INSTR("KUTYA","U"): REM =2
        PRINT INSTR("KUTYA","J"): REM =0
(ii)    PRINT INSTR("KUTYA","U",3): REM =0

(iii)   100 INPUT A$,B$,C$
        110 N=INSTR(A$,B$)
        120 IF N=0 THEN GOTO 100
        130 L1=LEN(B$)
        140 X$=LEFT$(A$,N-1)+C$+RIGHT$(A$,N+L1)
        150 PRINT X$
```

```
160 GOTO 100
```

Az (i) példa az INSTR hatását mutatja be. A (ii) példában használjuk az összehasonlítás kezdőpozíciójának beállítási lehetőségét. Ekkor a "KUTYA" sztringben az "U" már nem szerepel!

A (iii) alatti kis program az A$ sztringben a B$ szövegrészt a C$ szöveggel cseréli fel (feltéve, hogy megtalálja).

Hibalehetőségek A kifejezések kiértékelése közben számos hiba adódhat. Maga az INSTR hiba nélkül hajtódik végre.

INT

Rövidítés: nincs *Token:* $B5(181)

Mód: mind parancs, mind program módban használható.

Az argumentum egész részét számítja ki.

Szintaxis: INT(<aritmetikai kifejezés>)

Példák:

```
(i)   PRINT INT(X+.5)
(ii)  PRINT INT(123456.789): REM=123456
(iii) PRINT INT(-123456.789): REM=-123457
(iv)  35 INPUT X: IF X<>INT(X) THEN GOTO 35
```

A legtöbb kerekítési eljárás használja az INT függvényt. Az (i) sor egyszerű példát mutat a kerekítésre. Ha egy számot a legközelebbi egész számra akarunk felkerekíteni, akkor .5-et kell hozzáadni és utána az egész részét képezni.

A (ii) és (iii) példa azt szemlélteti, mi a különbség a pozitív és a negatív számok egész részének képzése között. Utolsó példánk azt ellenőrzi, hogy egész számot adtunk-e be inputként.

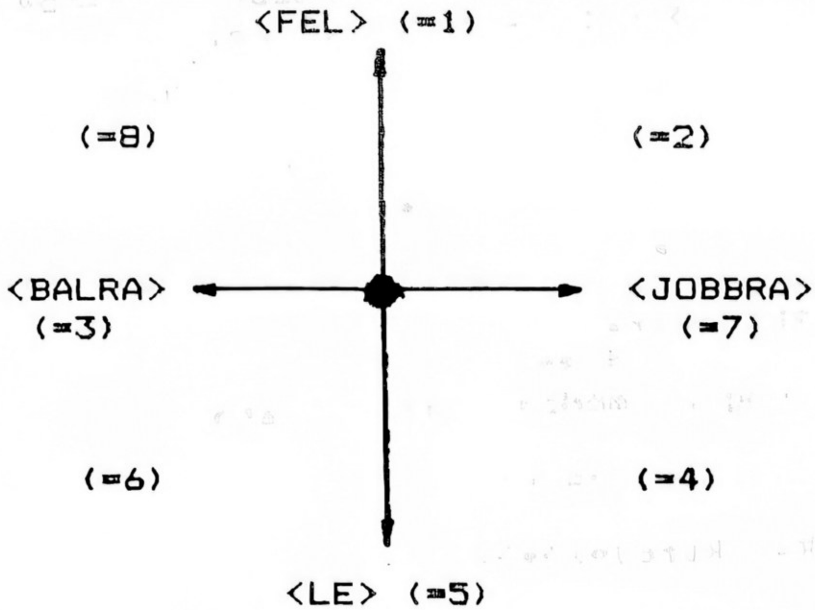
Hibalehetőségek Az argumentum kiértékelése közben számos hiba adódhat. Magának az INT-nek a kiszámítása már hiba nélkül történik.

JOY

Rövidítés: jO *Token:*$CF(207)

Mód: mind parancs, mind program módban használható.

Az argumentumként megadott botkormány (joystick) állapotával tér vissza. Az egyes értékek jelentése a következő:



<TÜZ> (=+128)

Szintaxis: JOY(<arit.kif.>)

Példák:

(i)
```
10 PRINT JOY(1): GOTO 10
```

(ii)
```
10 DATA "ESZAK",ESZAKKELET","KELET","DELKELET"
20 DATA "DEL","DELNYUGAT","NYUGAT","ESZAKNYUGAT"
30 FOR I=1 TO 8: READ IR$(I): NEXT I
40 DO
50 X=JOY(0): T=X AND 128: IR=X AND 127
60 IF IR<>0 THEN PRINT IR$(IR)
70 IF T<>0 THEN PRINT "TUZ!!!!": ELSE PRINT
80 LOOP
```

Az (i) program folyamatosan a képernyőre írja az első botkormány állapotát. A (ii) program ugyanazt a feladatot végzi el, csak szöveggel írja ki a botkormány állapotát.

Hibalehetőségek Az aritmetikai kifejezés kiértékelése közben számos hiba adódhat. Ha értéke nem 1, vagy 2, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

KEY

Rövidítés: kE *Token:*$F9(249)

Mód: mind parancs, mind program módban használható.

Segítségével megváltoztathatjuk, illetve megnézhetjük az <f.> billentyűkhöz rendelt szövegrészeket.

Szintaxis:
KEY [<arit.kif.>,<sztring.kif.>]

Az aritmetikai kifejezés az <f.> billentyű számát, a sztring kifejezés a hozzárendelt sztringet adja meg. A parancs kiadása után a megfelelő <f.> billentyű lenyomása ekvivalens a megfelelő sztring begépelésével. Ha a paraméterek elmaradnak, akkor az interpreter a képernyőre listázza az <f.> billentyűkhöz rendelt szövegeket. A <HELP> billentyű a 8. <f.> billentyűnek felel meg. Ha jelentését módosítani szeretnénk, akkor ezzel a sorszámmal kell rá hivatkozni.

Példák:
(i) KEY
(ii) KEY 3,"GOSUB"
(iii) KEY 2,"RUN1000"+CHR$(13)

Az (i) alatti példa a képernyőre listázza az <f.> billentyűk jelentését. A (ii) példa az <f3> billentyűhöz rendeli a "GOSUB" karaktersorozatot. Erre olyan program esetén lehet szükség, amelyik sok önálló részből áll, amelyek külön-külön is használhatók. Ebben az esetben a GOSUB 1200-at így is beírhatjuk: <f3>1200<return>

A (iii) példa a sztringkifejezések szerepeltetését mutatja. Ez tipikus a <RETURN> billentyű hozzáírására. A (iii) alatti parancs kiadása után az <f2> lenyomása ekvivalens a RUN 1000 parancs kiadásával, s a program azonnal el is indul!

Hibalehetőségek A kifejezések kiértékelése közben számos hiba történhet. Ha az aritmetikai kifejezés értéke nem esik az 1-8 intervallumba, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

LEFT$

Rövidítés: leF (a $ jel már nem kell!) *Token:* $C8(200)

Mód: mind program, mind parancs módban használható.

LEFT$ kétváltozós sztringfüggvény, amelyik az első argumentumként megadott sztring baloldali karaktereiből egy új sztringet képez. Az ehhez felhasználandó karakterek számát a második argumentum adja meg. Legyen például

```
     X$="KOVACS JANOS"
Pozíció: 123456789012
```

Ekkor LEFT$(X$,6)="KOVACS"

Szintaxis: LEFT$(<sztring>, <aritmetikai kifejezés>)

Példák:

```
(i)   PRINT LEFT$("HOGY VAGY",4): REM="HOGY"
(ii)  PRINT LEFT$("HOGY VAGY",123): REM NEM VALTOZIK
(iii) PRINT LEFT$(X$+"                    ",15)
      PRINT LEFT$(STR$(L)+"                    ",15):
      PRINT L;LEFT$("                    ",15-LEN(STR$(L)))
```

(i) a LEFT$ hatását mutatja be. A második példa azt szemlélteti, hogy ha az aritmetikai kifejezés meghaladja a sztring hosszát, akkor eredményül az eredeti sztringet kapjuk (minden hibajelzés nélkül). (iii) alatt példákat adtunk arra vonatkozólag, hogyan egészíthetünk ki egy-egy sztringet szóközökkel éppen 15 karakterre. A példa első sorában ez egy 'valódi' sztring (X$), a következő kettőben az L valós változó karakteres alakja STR$(l).

LEFT$(X$,N) mindig helyettesíthető a MID$(X$,1,N) kifejezéssel.

Hibalehetőségek A paraméterek kiértékelése közben előforduló hibákon túl az N=0 érték ?ILLEGAL QUANTITY ERROR hibajelzést okoz.

LEN

Rövidítés: nincs *Token:* $C3(195)

Mód: mind parancs, mind program módban használhatü.

LEN aritmetikai függvény; az argumentum sztring **hosszát** adja meg.

Szintaxis: LEN(<sztring kifejezés>)

Példák:

```
(i)    10 PRINT LEN("HAHO OCSI!"):REM=10
(ii)   20 X$="ELJEN":PRINT LEN(X$+"MAJUS")+2
(iii)  312 FOR J=1 TO 10
       314 PRINT SPC(22/2-LEN(UZENET$(J))/2);UZENET$(J)
       316 NEXT J
(iv)   500 IF LEN(BE$)<>13 THEN PRINT "** HIBA **":GOTO 3456
(v)    780 X$="*#!%^@"
       790 FOR J=1 TO LEN(X$)
       800 IF G$=MID$(X$,J,1) THEN RETURN
       810 NEXT:PRINT "** FELISMERHETETLEN **"
```

Az első két példa a függvény hatását szemlélteti. A (iii) alatti néhány soros rutin az UZENET$(J) sztringeket írja ki a képernyő közepére. Harmadik példánk egy input rutin része, amelyik ellenőrzi, hogy a bevitt sztring az előírt hosszúságú-e. Az (v) példában egy egyszerű programot írtunk, amelyik ellenőrzi, hogy G$ egy előre megadott karakterkészlet eleme-e vagy sem. A ciklus végértékének természetesen 5-öt is írhattunk volna, de a program módositása ekkor nehezebb lenne.

Hibalehetőségek A sztring-kifejezés kiértékelése után egy legfeljebb 255 hosszúságú sztringet kell kapnunk, ebben az esetben LEN már hiba nélkül végrehajtódik.

LET

Rövidítés: lE *Token:* $88(136)

Mód: mind parancs, mind program módban használhatü.

Szintaxis:

[LET] {
<egész változó> = <aritmetikai kifejezés>
<valós változó> = <aritmetikai kifejezés>
<sztring változó> = <sztring kifejezés>
}

értékadó utasítás. A változók egyszerű és tömb változók egyaránt lehetnek. A LET szó kiírása **nem kötelező.** Ha egy utasitás első karaktere nem token, akkor az interpreter LET-et tételez fel.

Példák:

```
(i)   LET X=1234: X$="QWE": LET X%=12.45
(ii)  20 FOR J=1 TO 10: READ X%: LET A%(J)=X%:NEXT
(iii) 30 IF A+B>C THEN D%=A/B
```

Az (i) alatti parancs a különböző típusú változók használatát mutatja be. Egész változó esetén az aritmetikai kifejezés értékének egész része lesz a változó új értéke. Igy a parancs végrehajtása után X%=12. A (ii) példában egy összetett értékadás szerepel, egy ciklus a DATA sorokból olvas be, majd ezek értékét egy mátrixba írja. Utolsó példánk egy feltételes értékadást mutat be.

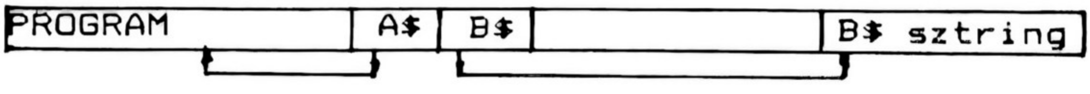
Néhány mikrogép esetén kötelező a LET használata (ilyen például a ZX81).

A változók értékei akárhányszor és a parancs vagy a program bármely részéből megváltoztathatók. A BASIC nem ismeri a 'lokális' és a 'globális' változók fogalmát. Ez különösen az alprogramok megtervezésekor okozhat problémát. Ha egy rutint a program több részéből is meghívunk, akkor változóit más célokra már nem célszerű használni.

A C-16 BASIC a sztringeket kétféleképpen tárolja. A két tárolási mód közti különbség néha fontos lehet. Tekintsük a következő programrészt:

```
10 A$="HELLO":B$="HELLO"+""
```

Az A$ illetve B$ értékét a program eltérően tárolja:



A változók területén a sztring neve, hossza, illetve a sztring első elemére mutató 2 byte kerül tárolásra. Amennyiben egy sztring-konstans kerül a változóba, a mutató magába a **programba mutat vissza** (helykímélés céljából). A sztringkifejezések eredményeiként előálló sztringek a BASIC munkaterületen kerülnek tárolásra. Ha most egy új programot töltünk be a PROGRAM-ból, akkor A$ értéke elvész, míg B$ értéke megmarad.

Hibalehetőségek Amennyiben a változó és a kifejezés típusa nem felel meg egymásnak, ?TYPE MISMATCH ERROR hibajelzést kapunk. Ha a változó, amelynek értékét a kifejezés definiálja, tömbváltozó, és indexei nem esnek a megfelelő értékhatárok közé, akkor ?BAD SUBSCRIPT ERROR hibajelzést kapunk. A kifejezés kiértékelése közben további hibaüzeneteket is kaphatunk.

Ha egy egész változónak adunk értéket, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kaphatunk, feltéve, hogy a kifejezés értéke nem esik a megengedett határok közé.

LIST

Rövidítés: lI *Token*:$9B(155)

Mód: mind program, mind parancs módban használható.

A BASIC munkaterületen tárolt programot, illetve annak a LIST paramétereiben specifikált részét írja ki a képernyőre (pontosabban az elsődleges outputra).

Szintaxis: LIST [<sorszám>] [- [<sorszám>]]

Példák:

```
(i)   LIST: REM AZ EGESZ PROGRAMOT KILISTAZZA
(ii)  LIST 5: REM CSAK AZ 5. SORT IRJA KI
(iii) LIST 5-135: REM AZ 5-TOL ES 135-IG ESO SOROKAT IRJA KI
      REM BELEERTVE 5-T ES 135-T IS
(iv)  LIST -135: REM A 135. SORIG LISTAZ
(v)   LIST 135-: REM A 135. SORTOL LISTAZ
(vi)  10 PRINT "** HIBA **":LIST 456
```

A példák a LIST paramétereinek használatát mutatják be. Normál körülmények között a programlista a képernyőn jelenik meg. Ha azonban az elsődleges outputot a CMD-vel megváltoztattuk, akkor a programlista a CMD utasításban specifikált file-ba íródik. Az **OPEN 3,4: CMD 3: LIST** parancs a programot a sornyomtatóra listázza ki. A program kazettára vagy lemezre is kilistázható.

(Ekkor azonban LOAD-dal már nem tölthető vissza!)

Az utolsó példa mutatja, hogy a LIST program módban is használható; hatása ugyanaz mint a STOP utasításé, azzal a különbséggel, hogy a program futása a CONT-tal nem folytatható, és a LIST paramétereinek megfelelő programsorokat az interpreter kilistázza.

A program listája nem betű szerint azonos azzal, ahogyan beírtuk az egyes programsorokat. A ? X utasítás a listán PRINT X alakban jelenik meg. A REM alapszó utáni grafikus jeleket (hacsak nem idézőjelben szerepelnek) a listázó rutin tokeneknek tekinti és a nekik megfelelő alapszavakat írja ki. A megfelelő memóriahelyek tartalmának módosításával elérhetjük, hogy a megjegyzés sor képernyővezérlő karaktereket is tartalmazzon. Ilyenkor a LIST hatására igen furcsa kiírási képet kaphatunk. Hasonló a probléma a sztring konstansokban szereplő vezérlő karakterekkel is. Gépeljük be a 10 ?"GO AWAY" utasítást, listázzuk ki a sort, és álljunk a kurzorral a második idézőjel fölé, szúrjunk be az <INS> billentyű segítségével nyolc szóközt, majd ezt töltsük fel <DEL> jelekkel (inverz T-ként jelennek meg). A listázás eredménye ezután: 10 PRINT.

A leghosszabb sor, amit ki tudunk listázni egy öt decimális jegyű sorszámból és 251 RESTORE tokenből áll. (Ezt a sort a szokásos módon nyilván nem tudjuk beírni a programba.)

Hibalehetőségek Gyakorlatilag nincsenek. A paramétereknek **nem kell létező** sorszámoknak lenniük, a LIST 100- utasítás az első, 100-nál nem kisebb számu sortól kezdi a program listázását.

LOAD

Rövidítés: 10 *Token:*$93(147)

Mód: mind program, mind parancs módban használható.
Az utasítás a paramétereiben megadott nevű programot az adott egységről és az adott módon betölti.

Szintaxis:
LOAD [<sztring kif.>] [,<arit. kif.>] [,<arit. kif.>]

Valamennyi paraméter opcionális, hiszen a kazettás egységen az első file egyértelműen azonosítható. A <sztring kifejezés> a **program neve.** Az első <aritmetikai kifejezés> az **egységszám**; ha elmarad, akkor a LOAD mindig a kazettás egységre vonatkozik (egységszám=1). A második <aritmetikai kifejezés> a töltés módját határozza meg (secondary adress). Kazettás file-ok esetén semmilyen hatást sem gyakorol.

Lemezes file-ok esetén az egységszám (8-11) mindig kötelező; ilyenkor a harmadik paraméter jelentése a következő:

0 a töltés a BASIC munkaterület elejéről kezdődik;
1 a töltés a program-file első két byte-ja által meghatározott címtől kezdődik.

Lemezes file-ok esetén a névben szereplő bizonyos karaktereknek speciális jelentése van (lásd az 5.fejezetet!). Ha a név *-gal végződik, például PROG*, akkor az utasítás a lemez katalógusában szereplő és "PROG"-gal kezdődő nevű *első* file-t tölti be. A file-névben szereplő kérdőjelek (?) tetszőleges karaktert jelentenek. LOAD "HE??O",8 például az első pontosan öt karakteres nevű file-t tölti be, ami nevének első két karaktere HE, ötödik karaktere pedig O.

Példák:

```
(i)    LOAD :REM A SZALAGROL AZ ELSO PROGRAM
       LOAD "PROG":REM A SZALAGROL A PROG NEVU PROGRAM
       LOAD "*",8 :REM LEMEZROL A KATALOGUS ELSO PROGRAMJA
       LOAD "A$*",8 :REM LEMEZROL AZ ELSO A$-RAL KEZDODOT
(ii)   90 LOAD "PROG",8
(iii)  45 PRINT "VARJ, TOLTOK!":LOAD "PROG2",8
```

Utolsó két példánk BASIC programból használja a LOAD utasítást. Program módban a LOAD utasítás használata eltér a parancs módban való használattól. A töltés befejezése után a program futása *elölről* kezdődik. Amennyiben BASIC programot töltöttünk be, az új, éppen betöltött program kezd el futni. Ezt a lehetőséget overlay technika kialakítására is felhasználhatjuk. A betöltés az eredeti program változóit nem törli, feltéve, hogy a másodszorra betöltött program rövidebb az elsőnél. A sztring

konstansokkal végrehajtott értékadások eredményei és a függvény definíciók azonban mindenképpen elvesznek.

Kazettás file esetén a következő információk kerülnek a képernyőre:

LOAD "PROGRAM",1 <RETURN>
PRESS PLAY ON TAPE <PLAY>
OK
SEARCHING FOR PROGRAM
FOUND NEV
FOUND PROGRAM
LOADING PROGRAM
READY.

A 'PRESS PLAY ON TAPE' üzenet megjelenése után kell a PLAY gombot lenyomnunk. Az interpreter addig keres a szalagon, míg az utasításban specifikált file-t meg nem találja, vagy egy EOT fejlécet nem olvas.

Lemezes file esetén az információ kevesebb:

LOAD "PROGRAM",8 <RETURN>

SEARCHING FOR PROGRAM
LOADING PROGRAM
READY.

Hibalehetőségek Elsősorban kazettás egységről való töltés esetén hardver hibák is történhetnek. ?FILE NOT FOUND hibajelzést kapunk, ha a szóban forgó file nem szerepel a lemez katalógusában, illetve a kazettán az EOT jelzőn túl olvasott az interpreter. ?DEVICE NOT PRESENT hibajelzést kapunk, ha az egységszámnak megfelelő egység nincs a C-16-hoz csatlakoztatva és bekapcsolva (ez a hibajelzés a kazettás egységgel kapcsolatban nem fordulhat elő). Gépi kódú programok töltése esetén a harmadik paraméter általában kötelező, s elhagyása hibát okozhat. Egy több gépi kódú programból álló program töltőprogramját mutatjuk be végül; az A értékének változtatása nélkül a program egy végtelen ciklusban mindig csak az első programot töltené be:

```
10 IF A=0 THEN A=1: LOAD"SKOT1",8,1
20 IF A=1 THEN A=2: LOAD"SKOT2",8,1
30 IF A=2 THEN A=3: LOAD"SKOT3",8,1
40 IF A=3 THEN A=4: LOAD"SKOT4",8,1
50 SYS4960
```

LOCATE

Rövidítés: loC *Token:*$E6(230)

Mód: mind parancs, mind program módban használható.

A grafikus kurzor helyzetét állíthatjuk be. A nagyfelbontású képernyő tartalma nem változik meg. A grafikus kurzor pillanatnyi helyzetét az RDOT(0),RDOT(1) koordinátapár adja meg.

Szintaxis: LOCATE <arit.kif.1>,<arit.kif.2>

Az első aritmetikai kifejezés az x, a második az y koordináta értékét határozza meg.

Példák:

(i) LOCATE 160,100
(ii) LOCATE 80,100

Az (i) példa a grafikus kurzort a normál, nagyfelbontású képernyő közepére állítja. A (ii) ugyanezt teszi többszínű, nagyfelbontású üzemmód esetén.

Hibalehetőségek Az aritmetikai kifejezések kiértékelése közben számos hiba adódhat. Ha a koordináta értékek egy nem létező pontra mutatnak, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

LOG

Rövidítés: nincs *Token:* $BC(188)

Mód: mind parancs, mind program módban használható.
Egyváltozós aritmetikai függvény, amelyik argumentumának e alapú logaritmusát számítja ki. Az EXP függvény inverze. (Lásd a (iv) példát.)

Szintaxis: LOG(<aritmetikai kifejezés>)

Példák:

```
(i)   PRINT LOG(10): REM =2.3026
(ii)  PRINT LOG(2.718281): REM =1 (gyakorlatilag)
(iii) PRINT LOG(X)/LOG(2): REM 2 ALAPU LOGARITMUS
(iv)  PRINT LOG(EXP(X)): REM =X
(v)   50 DEF FN LG(X)=LOG(X)/LOG(10):REM 10 ALAPU LOGARITMUS
      60 PRINT FN LG(100): REM =2
```

Példáink a LOG függvény használatának legegyszerűbb eseteit mutatják be. Magára a függvényre statisztikai, tudományos számítások során van szükség.

A LOG kiszámítása az előjel ellenőrzésével kezdődik, csak **pozitív számok**nak van logaritmusa. A LOG kiszámítása hatványsorral történik, amelynek összesen 4 (!) tagja van.

Hibalehetőségek Ha az argumentum értéke nem pozitív ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

LOOP

Rövidítés: loO *Token*:$EC(236)

Mód: mind parancs, mind program módban használható.

A DO ciklusok záró utasítása. A LOOP-ot követő kilépési feltétel értékétől függően vagy a LOOP-ot követő első utasítás, vagy a LOOP-hoz tartozó DO utasítás hajtódik végre.

Szintaxis:

```
          ⎧UNTIL⎫
LOOP [    ⎨     ⎬<logikai kif.> ]
          ⎩WHILE⎭
```

Ha a LOOP utasítás után nem áll sem UNTIL, sem WHILE, akkor a vezérlés mindig visszakerül a megfelelő DO utasításra. Az UNTIL esetében a logikai kifejezésnek igaznak, a WHILE esetében pedig hamisnak kell lennie ahhoz, hogy a ciklus lejárjon. Ellenkező esetben a vezérlés mindig a megfelelő DO utasításra tér vissza.

Példák:

```
(i)    DO: PRINT "I LOVE COMMODORE!": LOOP
(ii)   10 INPUT A%,B%
       20 DO
       30 R%=A%
       40 A%=B%-INT(B%/A%)*A%: B%=R%
       50 LOOP WHILE A%<>0
       60 PRINT "LNKO=";R%
(iii)  1200 DO
       1210 LOOP WHILE (JOY(1) AND128)=0
```

(i) a LOOP parancs módban való használatát mutatja. Az interpreter egy végtelen ciklusban folyamatosan újra és újra a képernyőre írja az "I LOVE COMMODORE" szöveget.

A (ii) program az A% és B% egész számok legnagyobb közös osztóját számítja ki az euklideszi algoritmus segítségével. A kilépés feltétele, hogy a maradék 0 legyen.

A (iii) példában addig 'jár' a ciklus, míg az 1-ás botkormány <TŰZ> gombját meg nem nyomjuk.

Hibalehetőségek A logikai kifejezés kiértékelése közben számos hiba adódhat. Ha az interpreter nem találja meg a LOOP-hoz tartozó DO-t, akkor ?LOOP WITHOUT DO ERROR hibaüzenetet kapunk. Ennek oka lehet egy nem megfelelő helyen végrehajtott LOOP, NEXT vagy RETURN utasítás is.

MID$

Rövidítés: mI(a $ jel már nem kell!) *Token:*$CA(202)

Mód: mind parancs, mind program módban használható.

Két-, vagy háromváltozós sztring-függvény, amelyik az első paraméterként megadott sztring bizonyos egymás után következő karaktereiből egy új sztringet állít elő.

*Szintaxis:*MID$(<sztring kif.>,<arit. kif.> [,<arit. kif.>]).

Az aritmetikai kifejezések értékei nem lehetnek 255-nél nagyobbak. A második paraméter a sztring azon karakterhelyét adja meg, **ahonnan** az új sztring kezdődik. A harmadik paraméter (ha van) az új sztring **hosszát** (az átmásolandó karakterek számát) adja meg. Ha nincs ennyi karakter a paraméterként szereplő sztringben, akkor a sztring végéig másolódnak át a karakterek. Ha a harmadik paraméter hiányzik, akkor az 255-nek felel meg.

```
Legyen X$="KOVACS PETER".
           123456789012

Ekkor MID$(X$, 3,6)="VACS P"
      MID$(X$, 8,10)="PETER"=MID$(X$,8).
```

Példák:

```
(i)  N$=MID$(STR$(N),2)
(ii) MO$=MID$("JANFEBMARAPRMAJJUNJULAUGSEPOCTNOVDEC",3*M-2,3)
```

Az (i) példánk a STR$(N) első karakterét törli. Ha például N=23, akkor STR$(N)=" 23", de N$="23". Természetesen N=-23 esetén is N$="23" lesz!

A második példa azt mutatja be, hogyan lehet MID$-t 'tömbként' használni. Az utasítás végrehajtása után MO$ az M. hónap hárombetűs rövidítését tartalmazza.

A LEFT$ és a RIGHT$ függvények a MID$ segítségével kifejezhetők:

```
LEFT$(X$,N) = MID$(X$,1,N)
RIGHT$(X$,N)= MID$(X$,LEN(X$)-N+1)
```

Hibalehetőségek ?ILLEGAL QUANTITY ERROR hibajelzést kapunk, ha a sztring hossza, vagy a paraméterek nagyobbak 255-nél. Hasonló jelzést kapunk, ha az eredmény egy nullsztring lenne. Így például MID$("",I,J) mindig hibás lesz.

MONITOR

Rövidítés: mO *Token:*$FA(250)

Mód: csak parancs módban használható.

Belépés a gépi kódu monitorba.

Szintaxis: MONITOR

A parancs kiadásával a C-16 beépített gépi kódú monitorába lépünk be. A BASIC utasításokat a kilépésig nem használhatjuk. Erre az <X> monitor parancs szolgál. A gépi kódú monitorról részletesen a 8. fejezetben szólunk.

NEW

Rövidítés: nincs *Token:*$A2(162)

Mód: mind parancs, mind program módban használható.

Törli a memóriában tárolt BASIC programot és a változókat.

*Szintaxis:*NEW

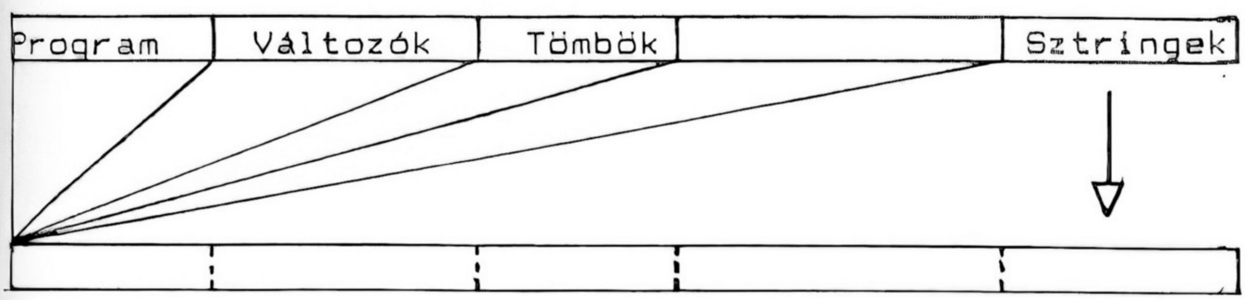
A NEW utasításnak nincsenek paraméterei.

Példák:

```
(i)  NEW
(ii) 1000 PRINT "ODA AZ EGESZ":NEW
```

Mind parancs módban, mind program módban a hatás ugyanaz. A program és a változók törlődnek.

Az utasítás valójában *nem* törli a memória tartalmát, csupán a BASIC munkaterülethez kapcsolódó mutatókat állítja kezdőértékükre, ahogy az alábbi ábrán is látható:



Az interpreter ezek után úgy érzékeli, hogy üres a memória.

Hibalehetőségek Nincs (de elveszhet a program).

NEXT

Rövidítés: nE *Token:* $82(130)

Mód: mind parancs, mind program módban használható.

Az utasítás hatására a vezérlés a NEXT-nek megfelelő FOR-t követő első utasításra kerül; feltéve, hogy a STEP-ben specifikált értékkel megnövelt ciklusváltozó még mindig a megengedett értéktartományba esik. Ha nem, a NEXT-et követő első utasítás hajtódik végre.

Szintaxis: NEXT <változólista>

A változólista egymástól vesszővel elválasztott egyszerű, valós változókat tartalmazhat. NEXT I,J és NEXT I:NEXT J egymással ekvivalensek. Amennyiben a NEXT nem tartalmaz további paramétereket, akkor a legutoljára végrehajtott FOR utasítás alá tér vissza a vezérlés. Példákat a FOR utasítás leírásakor adtunk.

A FOR utasítás szintaxisát a FOR utasítás végrehajtásakor ellenőrzi az interpreter. Az összes szükséges paraméter értéke (összesen 18 byte) a verembe kerül; legvégén a FOR tokenje: 129. A NEXT utasítás végrehajtása a ciklusváltozónak megfelelő 18 byte megkeresésével kezdődik. Ha az interpreter nem találja meg, akkor ?NEXT WITHOUT FOR hibajelzést kapunk. Ha megtalálta a megfelelő FOR tokent, akkor a ciklusváltozó értékét megnöveli a lépésközzel. Ha a ciklusváltozó új értéke átlépi a FOR utasításban megadott végértéket, akkor a NEXT utáni utasításra kerül a vezérlés. Ha nem, akkor a FOR utáni utasításra.

Hibalehetőségek A NEXT I utasítás *törli* a veremből a felesleges NEXT-eket. Helytelenül struktúrált programok esetén ez később ?NEXT WITHOUT FOR hibajelzést eredményezhet.

NOT

Rövidítés: nO *Token:* $A8(168)

Mód: mind parancs, mind program módban használható.

Egyváltozós logikai függvény. Az argumentumot nem kell zárójelbe tenni.

Szintaxis: NOT <aritmetikai kifejezés>

Példák:

```
(i)   105 IF PEEK(X)=34 THEN ID = NOT ID
(ii)  PRINT NOT 23456: REM =-23457
(iii) 300 IF NOT OK THEN PRINT "** HIBA **":END
```

Az első sor egy listázó rutinban szerepel. Ha a következő byte egy idézőjel ("), akkor az idézőjel jelenlétét ellenőrző ID változó (logikai) ellentettjére változik.

A második példa a NOT használatát aritmetikai argumentum esetén mutatja be. 23456=$5BA0, így a NOT műveletet *bitenként* elvégezve a $A45F hexadecimális számot kapjuk, amelyik a -23457 számnak felel meg. A nyomtatás eredményeként tehát ennyit kell kapnunk.

Utolsó példánkban a NOT használatát feltételes vezérlő utasításban mutatjuk be. Ha az OK változó nem igaz (azaz 0), akkor a HIBA szöveg kiíródik a képernyőre, és a program megáll.

Hibalehetőségek A NOT művelet a logikai műveletek közt a legmagasabb rangú, ezért először a NOT hajtódik végre. Ennek megfelelően tartsuk szem előtt, hogy például a NOT A AND B ezt jelenti: (NOT A) AND B. A NOT után álló kifejezés értékének a [-32768,32767] intervallumba kell esnie. Különben ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

ON

Rövidítés: nincs *Token:* $91(145)

Mód: mind parancs, mind program módban használható.

Többirányú elágazást tesz lehetővé.

Szintaxis:

ON <aritmetikai kifejezés> {GOSUB | GOTO} <sorszámlista>

A <sorszámlista> egymástól vesszővel elválasztott előjel nélküli egész konstansokat tartalmaz. Az aritmetikai kifejezést értékeli ki először az interpreter (az értéknek 0-255 intervallumba kell esnie), majd a lista annyiadik elemének megfelelő programsorra adódik át a program vezérlése. GOSUB esetén a verembe még a visszatéréshez szükséges információk is bekerülnek.

Példák:

```
(i)   1000 ON SGN(X)+2 GOTO 2000,3000,4000
(ii)  60 ON 1+RND(1)*10 GOTO 10,20,30,40,50,60,70,80,90,100
(iii) 200 ON Q GOSUB 150,200,250
```

Az (i) példa egy előjel szerinti elágazást mutat. Ha X<0 a 2000., ha X=0 akkor a 3000., ha pedig végül X>0, akkor a 4000. sorral folytatódik a program. (ii) egy véletlen programelágazást mutat be, ilyen programrészek főleg játékprogramokban fordulnak elő. A vezérlés - véletlenszerűen - a 100., 200. stb programsorok egyikére adódik.

Az interpreter nem ad hibajelzést, ha például a sorszámlista három értéket tartalmaz, de az aritmetikai kifejezés értéke 4. Ha a (iii) példában Q értéke 0 vagy 4, nem kapunk hibajelzést, a program futása a **következő** sorral folytatódik.

Hibalehetőségek ?ILLEGAL QUANTITY hibajelzést kapunk, ha az aritmetikai kifejezés egész része nem esik a 0-255 intervallumba. ?SYNTAX ERROR üzenetet kapunk, ha az aritmetikai kifejezést nem GOTO vagy GOSUB token követi. A GO TO alak használata nem megengedett! Végül ha a szóban forgó számú sor nem létezik, akkor ?UNDEF'D STATEMENT hibajelzést kapunk.

OPEN

Rövidítés: oP *Token:* $9F(159)

Mód: mind parancs, mind program módban használható.

Az utasítás első paramétereként szereplő számot, mint logikai file számot felveszi egy táblázatba az esetleges egységszámmal, illetve megnyitási móddal együtt. Amikor egy BASIC utasítás, például egy PRINT# erre a logikai file számra hivatkozik, a hozzá tartozó egységszám és megnyitási mód paramétereket a táblázatból előkeresi az interpreter, és ezek értékének megfelelően hajtja végre az illető utasítást. Az OPEN utasítás hatására azoknál az egységeknél, ahol ez szükséges, az egységen fizikai értelemben is megnyilik a file. Kazettás file-ok esetén ez a cimke írását/olvasását jelenti. Lemezes file-ok esetén a paraméterek parancsként a soros buszra kerülnek, az egység megnyitja a file-t, és egy puffert köt le a file-lal való adatcserére.

Szintaxis:
OPEN <arit.kif.> [,<arit.kif.>][,<arit.kif.>] [,<sztring kif.>]

Az első paraméter kötelező. értéke az 1-255 intervallumba kell hogy essen. Ez a paraméter a **logikai file szám**. A második kifejezés az egység **hardver szám**át adja, ez a 0-15 intervallumba eshet. A harmadik kifejezés a **megnyitási mód**, értéke ugyancsak a 0-15 intervallumba esik. A <sztring kifejezés> a file nevét és egyéb jellemzőit adja meg.

Példák:
```
(i)   100 OPEN 10:REM=OPEN 10,1,0
(ii)  110 OPEN 15,8,15 :REM A HIBACSATORNA MEGNYITASA
      120 OPEN 1,8,4,"$":REM A # PSZEUDO-FILE MEGNYITASA
      130 OPEN 2,8,2,"FILE,S,W":REM A FILE MEGNYITASA IRASRA
(iii) 200 OPEN D,D
      220 <AZ EREDMENYEK KIIRASA>
      230 CLOSE D
```

Az első példa egy kazettás file-t nyit meg. (Mivel a kazettás file megnyitása a file fejlécének (header) olvasásával vagy írásával jár együtt, ezért kazettán egyszerre csak egy file lehet megnyitva.) A fenti utasítás hatására a fejléc a kazetta-pufferbe kerül és ott (pl. PEEK-kel) megvizsgálható. A második példában a 8-as lemezegységen három file-t (15,1,2,3 logikai file-számokkal) nyitunk meg. A harmadik példában D értékétől függően (0,4) vagy a képernyőt, vagy a nyomtatót nyitjuk meg. Az eredmények kiírására csak olyan szerkesztő karaktereket használhatunk, amelyek mindkét egységen ugyanúgy hatnak.

A C-16 a következő megnyitási módokat értelmezi:

Periféria neve	száma	Megnyitási mód	Parancs
Billentyűzet	0		
Kazettás magnó	1	0=input 1=output 2=output+EOT	a file neve
Modem	2	0	kontrol regiszterek
Képernyő	3	0,1	
Nyomtató	4,5	0=grafikus jelek 7=normál	kiírandó szöveg
Lemezegység	8-11	2-14 adatcsatorna 15=hiba csatorna 0=SAVE 1=LOAD	meghajtó a file neve írás/olvasás file típusa

Az OPEN utasítás hatására az első három paraméter értéke bekerül a nyitott file-okat leíró táblázatokba. Ha a táblázatban már szerepel egy ilyen sorszámú logikai file, akkor ?FILE OPEN ERROR hibaüzenetet kapunk. Ha mind a 10 logikai file nyitott, akkor ?TOO MANY FILES ERROR hibaüzenetet kapunk.

Hibalehetőségek A paraméterek kiértékelése közben különféle hibajelzéseket kaphatunk. Programhiba után célszerű az összes file-t lezárni, különben újraindítás után ?FILE OPEN ERROR hibajelzést kaphatunk. Nem megfelelő működést eredményezhet a, megnyitási mód, illetve a parancs sztring szerkezetének pontatlan ismerete.

OR

Rövidítés: nincs *Token*:$B0(176)

Mód: mind parancs, mind program módban használható.

Két aritmetikai kifejezés logikai 'vagy'-át számítja ki. Az OR műveléttáblája a következő:

OR	hamis	igaz
hamis	hamis	igaz
igaz	igaz	igaz

Szintaxis:
<aritmetikai kifejezés> OR <aritmetikai kifejezés>

Példák:

```
(i)   100 IF A<0 OR A>2000 THEN A$="ROSSZ"
(ii)  200 PRINT -1 OR 1234: REM =-1
(iii) 210 PRINT 380 OR 75: REM =383
(iv)  400 A%=12+(A<0 OR A>7)*(X-9)
```

Az (i) és (iv) példában az OR tipikus használatát látjuk. Az első egy egyszerű feltételes értékadás. (iv)-ben a feltételes értékadás az aritmetikai kifejezésbe épül bele, felhasználva, hogy a logikai kifejezések egyben aritmetikai kifejezések is. A (ii),(iii) példák az OR aritmetikai kifejezésekkel való használatát mutatják be. A második sor eredménye -1, hiszen -1=$FFFF alakban kerül tárolásra, és itt mindegyik bit magas. Az OR művelet eredménye így $FFFF=-1 lesz. A harmadik példa a 00000001 01111111 bit-képet adja, ami 383.

Az 'OR' művelet a műveletek közt a legalacsonyabb rendű, így utoljára hajtódik végre.

Hibalehetőségek Megegyeznek az AND utasításnál leírtakkal.

PAINT

Rövidítés: pA *Token*: $DF(223)

Mód: mind parancs, mind program módban használható.

Lehetőséget nyújt arra, hogy a nagyfelbontású képernyő valamely zárt, folytonos vonallal körülhatárolt részét ***befessük***, ***besatírozzuk***. Ehhez az utasításban a festés színtípusán kívül a befestendő terület egy pontját kell megadnunk. A festés a nagyfelbontású képernyő tartalmától függ.

Szintaxis: PAINT [<színtípus>] [, [<x>,<y>] [,<mód>]]

A besatírozott pontok színe a <színtípusnak> megfelelő szín lesz. Ha elhagyjuk, akkor az 1-es színnel (az írás színe) dolgozik a gép. <x>,<y> a befestendő terület egy pontja, ha elmarad, akkor a grafikus kurzor aktuális helyén kezdődik a festés. A <mód> paraméter a határvonal értelmezését adja meg. Ha értéke 0, akkor a festés a bekapcsolt pontokig tart, ha pedig 1, akkor a háttérszínű pontokig. Mind a négy paraméter tetszőleges aritmetikai kifejezés lehet.

Példák:

(i)
```
10 GRAPHIC 1,1
20 CIRCLE ,160,100,60,60
30 PAINT ,160,100
```

(ii)
```
10 GRAPHIC 1,1
20 FOR I=0 TO 11
30 CIRCLE 1,159,100,20,80,,,15*I
40 NEXT I
60 FOR I=1 TO 6
65 FOR J=0 TO 1
70 PAINT J,0,0,J
80 NEXT J
90 NEXT I
100 GETKEY A$
110 GRAPHIC 0
```

Az (i) példa a PAINT egyszerű hatását mutatja be. A (ii) példában felváltva festünk az 1-es és a 0-ás színnel. Ennek hatására a képernyőn levő ábra végül eltűnik.

Hibalehetőségek Az aritmetikai kifejezések kiértékelése közben számos hiba adódhat. A <színtípus> értékének összhangban kell lennie a választott kijelzési móddal. Ha nem, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

PEEK

Rövidítés: pE *Token*:$C2(194)

Mód: mind parancs, mind program módban használható.

A PEEK egyváltozós aritmetikai függvény, amelyik az argumentumként megadott sorszámú memóriacímen levő byte értékével tér vissza.

Szintaxis: PEEK(<aritmetikai kifejezés>).

A kifejezés értékének a 0-65535 intervallumba kell esnie.

Példák:

```
(i)    PRINT PEEK(239)
(ii)   CIM=PEEK(43)+256*PEEK(44)
(iii   PRINT "<CLEAR>"; CHR$(34);:FOR J=CIM TO CIM+100:
           PRINT CHR$(PEEK(J));: NEXT
(iv)   200 X=DEC("FF06")
       210 POKE X,(PEEK(X) AND 240) OR 7
(v)    100 DEF FN DEEK(X)=PEEK(X)+256*PEEK(X+1)
       120 PRINT FN DEEK(43)
```

Az (i) példa a PEEK hatását mutatja be, a parancs kiadása kiírja a képernyőre asa 239. memória tartalmát. Ez általában 0, lévén, hogy az interpreter ebben a memóriában tárolja a billentyűzet-pufferben levő karakterek számát.

A (ii) példa a 43-44 címen levő két byte-os mutató értékét számitja ki. Ez a mutató adja meg, hol kezdődik a memóriában a BASIC program. A (iii) parancs (amelyet *egyetlen* sorba gépelve kell kiadni,) kiírja a képernyőre a BASIC programot. Természetesen nem a LIST-tel megszokott alakban, hanem abban a formában, ahogy a gép tárolja. Ezért kell az elején egy idézőjelet (CHR$(34)) kiírni, nehogy a vezérlő karakterek elrontsák a szöveget.

A (iv) példában a képernyőt függőleges irányban eltoljuk. Erre a $FF06 memóriacím alsó négy byte-ja szolgál. Ennek módositását úgy kell elvégeznünk, hogy a memória felső négy bitje ne változzon meg.

Az (v) példa egy *dupla-pontosságú* PEEK-et definiál, amelyik egy két byte-os mutató értékét számítja ki. Ezzel a függvénnyel a (ii) példában szereplő számítás a 120. sorban látható.

Hibalehetőségek Az aritmetikai kifejezésnek a 0-65535 intervallumba kell esnie. Ha nem, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

POKE

Rövidítés: pO *Token*:$97(151)

Mód: mind parancs, mind program módban használható.

Az utasítás segítségével egy tetszőleges memóriacímre közvetlenül beírhatunk egy 0-255 közti egész számot.

Szintaxis: POKE <aritmetikai kifejezés>,<aritmetikai kifejezés>

A két kifejezés a memória címét és a tárolandó byte-ot jelenti. Ennek megfelelően az első értékének a 0-65535, a második értékének pedig a 0-255 intervallumba kell esnie.

Példák:

```
(i)    10 FOR J=1 TO 255:POKE 3072+J,J:POKE 2048+J,6: NEXT J
(ii)   600 FOR J=4300 TO 9E9:POKE J,170
       610 IF PEEK(J)=170 THEN NEXT
(iii)  POKE DEC("FF11"),7
(iv)   1350 POKE 239,0
       1360 GETKEY A$: GRAPHIC 0
```

A három példában megpróbáltuk érzékeltetni azokat a lehetőségeket, amiket a POKE utasítás kínál.

Az első példa a képernyő tetejére kiírja mind a 256 karaktert. A (ii) példa a memória megadott részét ellenőrzi. Amikor a visszaolvasott érték már nem 170, az vagy a memória végét, vagy hibás címet jelent. (iii) a hanggenerátor **hangerejét** maximálisra állítja.

Utolsó példánkban az 1350. sorban látható POKE utasítás 'kiüríti' a billentyűzet-puffert. Ha a GETKEY végrehajtása előtt nyomtunk meg egy billentyűt, az most nem okozhatja, hogy véletlenül térjünk vissza karakteres kijelzésre.

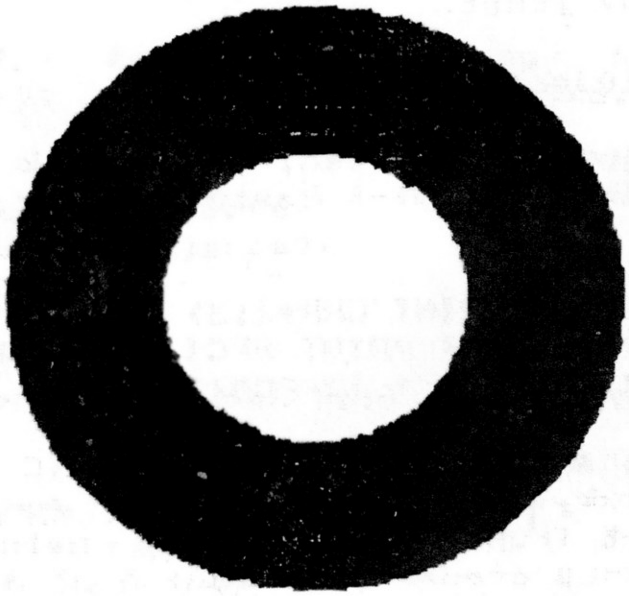
Egy 'dupla-pontosságú' POKE függvényként nem írható fel. Egy egyszerű parancs, amelyik ezt elvégzi, a következő:

```
POKE Z1,Z2-INT(Z2/256)*256:POKE Z1+1, Z2/256
```

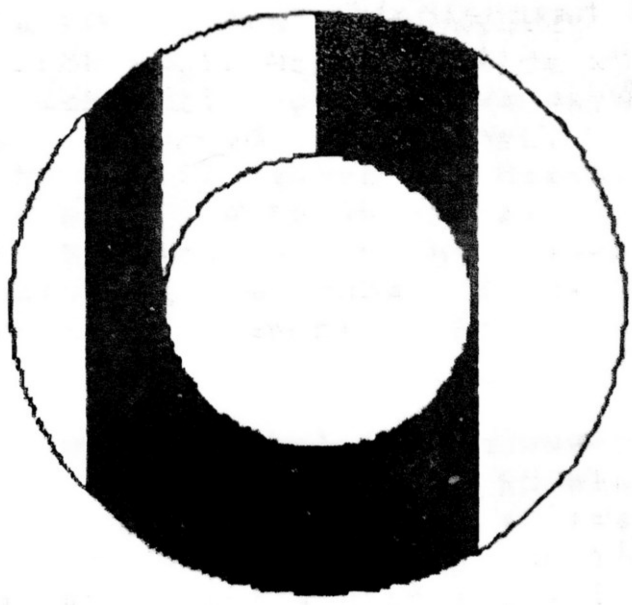
Hibalehetőségek A POKE paramétereinek kiértékelése közben számos hiba történhet. Ha a memóriacím, vagy a byte értéke nem megfelelő, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

Az alábbi két ábra a PAINT rutin működését szemlélteti. A rajzok a 233. oldalon levő *GYURU* nevű program futásáról készültek.

Az első ábra a program végeredményét mutatja. Ezen nem látni, hogy a PAINT milyen sorrendben *festi* be az egyes pontokat. A második ábrát úgy kaptuk, hogy a PAINT végrehajtása közben megnyomtuk a <STOP> billentyűt. Ennek hatására a program futása abbamaradt. A rajzon is látszik, hogy a PAINT először a jobb alsó pontokat festi be, s azután halad csak tovább.



A teljes ábra.



POS

Rövidítés: nincs *Token:* $B9(185)

Mód: mind parancs, mind program módban használható.

Kiszámítja a kurzor helyzetét az aktuális képernyő sorban. A lehetséges érték a 0-255 intervallumba esik. Ez nem a képernyő 40 karakteres sorában levő pozició, hanem annak a mértéke, amennyivel a kurzort a sor elejétől elmozgattuk. Programsorok esetén ez maximum 87 lehet.

Szintaxis: POS(<kifejezés>)

A POS-nak ál-argumentuma van, amelynek értéke, típusa lényegtelen. Általában POS(0)-t használunk.

Példák:

```
(i)   If POS(0)>20 THEN PRINT CHR$(13)
(ii)  PRINT TAB(10)POS(0);: PRINT SPC(10)POS(0)
(iii) PRINT LEFT$("            ",12-POS(1));K$
```

A POS talán a leghasznavehetetlenebb BASIC alapszó. Az (i) példában ellenőrizzük, hogy nem vagyunk-e a sor végén, s ha igen, egy <RETURN>-t írunk ki. A (ii) példa a POS hatását szemlélteti. A parancs eredményül végül 5-öt és 13-at nyomtat. A harmadik példa a kurzor helyzetétől a 12. pozícióig szóközöket nyomtat, utána pedig kiírja a K$ sztringet.

A POS ugyanazokat a paramétereket használja, mint a TAB. Ezért a POS utasítás nyomtatókkal kapcsolatban csak egy és ugyanazon **fizikai soron** belül használható.

Hibalehetőség Nincs.

PRINT

Rövidítés: ? *Token:* $99(153)

Mód: mind parancs, mind program módban használható.

Kiszámítja a paraméterek értékét, és az utasításban megjelölt formában kinyomtatja az elsődleges output file-ba (általában a képernyőre).

Szintaxis: PRINT[<nyomtatási kép>]

A <nyomtatási kép> aritmetikai és/vagy sztring kifejezések sorozata, amelyeket egymástól a következő **szeparátor**okkal választhatunk el:

SPC(<aritmetikai kifejezés>)
TAB(<aritmetikai kifejezés>)
szóköz
vessző (,)
pontosvessző (;)
(Ahol ez megengedett, nem kell szeparátor).

Példák:

```
(i)   FOR J=0 TO 255: PRINT CHR$(J);:NEXT
(ii)  FOR J=0 TO 100: PRINT J,:NEXT
(iii) PRINT X+Y;124;P*G*(1-V)
(iv)  PRINT "HELLO";A$B$U$U1$;MID$(X$,2)
(v)   PRINT TI;TI$;ST;<pi>
(vi)  PRINT: PRINT
```

A fenti hat példa a PRINT utasítás szinte valamennyi lehetőségét bemutatja, kivéve a SPC és TAB használatát (ezek a megfelelő kulcsszónál találhatók meg). Az első példa 256 karaktert ír ki a képernyőre. Mivel ezek közt vezérlő karakterek is vannak, a kiírás közben a képernyő törlődhet, színes karakterek jelenhetnek meg stb. (ii) a vessző (,) használatát mutatja be. A PRINT utasításban szereplő értékek kiírása a **legelső tabulálási** ponttól kezdődik. A 22 karakter hosszú sorban két tabulálási pont van az 1. illetve 12. oszlopban. Ennek megfelelően a (ii) példa a 100 számot 50 sorban írja ki, minden sorba kettőt-kettőt.

Ha a kinyomtatandó mennyiségeket pontosvesszővel (;) választjuk el egymástól, akkor a nyomtatás a **következő karakterhelyen** kezdődik. (iii) és (iv) példa az aritmetikai és sztring kifejezések használatát mutatja be. Az aritmetikai kifejezések értéke a STR$(X) alakban nyomtatódik ki, ami után még egy <CRSR JOBBRA> is kinyomódik. Ilyen módon még ha pontosvesszővel (;) elválasztott számokat is nyomtatunk ki, egy szóköz mindig kerül közéjük. Pozitív számok esetén a pozitív előjel helyett egy

szóköz nyomtatódik ki. A (iv) példa a szeparátorok elhagyását is bemutatja. A $, % és a tömbváltozók végét jelző záró zárójelet az interpreter **terminátor**nak tekinti, ezért ezek után a **pontosvessző elhagyható.**

A szeparátorok elhagyása több esetben is lehetséges, de ezeknek a használatát nem javasoljuk:

```
10 PRINT (2)(7) :REM 2  7
20 PRINT 1.2..3 :REM 1.2  0  .3
30 PRINT KESZ.  :REM 0  0
40 PRINT ;,L*K;4:REM             0  4
```

Az (v) példánk azt mutatja be, hogy a PRINT utasítás felismeri a fenntartott változókat (TI, TI$, ST) és a pi-t.

A (vi) példa az üres PRINT utasítás használatát mutatja be: hatására két soremelés nyomtatódik ki.

Vezérlő karakterek használata Mint már a 2. fejezetben említettük, a PRINT utasításban szereplő sztring kifejezések vezérlő karaktereket is tartalmazhatnak, ezek 'kinyomtatása' a megfelelő vezérlő funkció végrehajtásával jár. Ugyanezt a hatást a CHR$(X) alakú sztringek használatával is elérhetjük. Például a

```
PRINT CHR$(18)"HAHO" CHR$(146)"HAHO"
```

parancs az első HAHO-t inverz alakban írja ki. A képernyőn nem biztos, hogy megjelenik a PRINT utasítással kiírt szöveg, mert az, amit a képernyőn látunk, nemcsak a képernyő memória tartalmától függ. Részletesen a 6. fejezetben szólunk a C-16 grafikus lehetőségeiről.

Hibalehetőségek A SPC **és** TAB paraméterei egész részének a 0-255 intervallumba kell esnie. Amennyiben az utasításban nincs szintaktikus hiba, végrehajtódik. Ez persze messze nem jelenti azt, hogy olyan képernyőképet is kapunk, amilyent megterveztünk.

PRINT#

Rövidítés: pR (a # jel már nem kell!) *Token:*$98(152)

Mód: mind parancs, mind program módban használható.

Kiszámítja a paraméterek értékét és a megadott formában kiírja az utasításban specifikált logikai file-ba.

Szintaxis: PRINT# lf [,<nyomtatási kép>].
lf a logikai file szám. A <nyomtatási kép> alakja azonos a PRINT utasításban szereplővel. A PRINT és a # jel közé nem írhatunk szóközt!

Példák:

(i)
```
100 OPEN 4,4
110 PRINT#4,"ARAJANLAT:"
```

(ii)
```
200 OPEN 1,4,0: OPEN 2,4,7
210 PRINT#1,"ARAJANLAT";: PRINT#2,"ARAJANLAT"
```

(iii)
```
1000 OPEN 5,8,5,"ADATOK,S,W"
1010 FOR J=1 TO 10: READ A: PRINT#5,J;A: NEXT J
1020 CLOSE 5
1030 DATA 1,2,3,4,5,6,7,8,9,10
```

A fenti példák csak a nyomtató és a lemezegység használatát mutatják be, de hasonlóan lehet a többi perifériás egységet is programozni. Az (i) példa a nyomtatóra írja ki az "ARAJANLAT" szöveget.

A (ii) példában két csatornát nyitottunk meg a nyomtatón, és felváltva használjuk ezeket a PRINT# utasításban. Ennek hatására az egyik "ARAJANLAT" nagy-, a másik kisbetűkkel nyomtatódik ki. A #1 logikai file ugyanis a nagybetűk/grafikus jelek karakterkészlet, míg a #2 file a kisbetűk/nagybetűk karakterkészlet segítségével nyomtat.

Utolsó példánk az ADATOK nevű lemezes file-ba ír 10 számot.

Hibalehetőségek A SPC, a TAB és a vezérlő karakterek a kiírás során nehézségeket okozhatnak; előfordulhat, hogy a file tartalma nem egyezik meg a kívánttal.

PRINT USING

Rövidítés: ?usI *Token:* USING $F3(251)

Mód: mind parancs, mind program módban használható.

Lehetővé teszi számok és sztringek formázott kiírását. A formátumot megadó sztringben a következő - a kiírás formátumát vezérlő - karakterek szerepelhetnek:

Karakter	**Jelentés**
#	értékes számjegy vagy karakter
+	a pozitív előjel kiíratása
-	szám előjelének helye
.	tizedespont helye
,	elválasztó jel
$	dollárjel helye
^^^^	exponens helye
=	középre igazítás
>	jobbra igazítás

Szintaxis: PRINT USING <formátum>;<kifejezés **lista**>

A <formátum> tetszőleges sztringkifejezés lehet. Ennek a karakterei határozzák meg a kifejezések kiíratásának formáját. A <kifejezés lista> tetszőleges kifejezések **vesszővel elválasztott** sorozata.

A C-16 interpreter párhuzamosan dolgozza fel a <formátum> sztringet és a <kifejezés listát>. Ha a kettőt nem találja összeegyeztethetőnek, akkor a feldolgozást abbahagyja, és ?SYNTAX ERROR hibaüzenettel abbamarad a program futása. Még ha a kiíratási formátum értelmezhető is, előfordulhat, hogy az előírásnak megfelelően nem jeleníthető meg a szóban forgó mennyiség. Sztringek esetén ilyenkor a sztring néhány utolsó karaktere elvész. Számok esetén az értékes jegyek és a tizedespont helyen *-ok nyomtatódnak ki.

Ha a <formátum> előbb merül ki, mint a <kifejezés lista>, akkor az interpreter újra kezdi a <formátum> feldolgozását.

Példák:

```
(i)   100 DEF FN G(X)=4*X^2-6*X+2
      110 FOR I=1 TO 60
      120 PRINT USING "   ####.##";FN G(I)
      130 NEXT I
(ii)  120 PRINT USING "   ####.##^^^^"; FN G(X)
      120 PRINT USING "   #.#####^^^^"; FN G(X)
```

```
(iii) 1000 A$="Commodore 16"
      1010 PRINT USING "#####################";A$
      1020 PRINT USING "####################>";A$
      1030 PRINT USING "####################=";A$
```

(i)-ben a 100. sorban definiált G(X) függvény helyettesítési értékeit listázzuk ki. Az utolsó számok helyett már csillagok jelennek meg, mert a kiírandó szám a kívánt módon már nem írható ki (túl nagy). (ii)-ben szereplő két 120-as sor az első példa azonos sorának a módosítása. Az exponenciális kijelzés miatt itt már nem történik túlcsordulás. A második fajta kijelzés felel meg a zsebszámítógépeken gyakran szereplő lebegőpontos kijelzési módnak.

A (iii) példa a sztring kifejezések kiíratásánál használható =, illetve > karakterek szerepét mutatja. Normál esetben a sztring balra igazítva íródik ki. Az '=' használata esetén középre, a '>' használata esetén pedig jobbra igazítva. Ha a kiírandó sztring hosszabb, mint a mezőszélesség, akkor a sztring utolsó karakterei elvesznek. A mezőszélességbe a # jeleken kívül az = és a > jel is beleszámít.

Hibalehetőségek A kifejezések kiértékelése közben számos hiba adódhat. A <formátum> sztring karaktereinek típusban és alakban meg kell felelnie a kifejezések típusának. Ha ez nem teljesül, akkor ?SYNTAX ERROR hibajelzést kapunk.

PUDEF

Rövidítés: pU *Token*:$DC(221)

Mód: mind parancs, mind program módban használható.

Lehetőséget ad a PRINT USING-ban használt egyes jelek átdefiniálására.

Szintaxis: PUDEF <sztring kifejezés>

A <sztring kifejezés> értékeként kapott sztring maximum négy karakterből áll. Ezek a PRINT USING-ban használt következő karaktereknek felel meg:

Sorszám	**Jelentés**
1	elválasztó jel
2	vessző
3	tizedespont
4	$ jel

Példák:

```
(i)   PUDEF " ;,"
(ii)  PUDEF "*,.@"
```

Az (i) parancs kiadása után a PRINT USING tizedespont helyett tizedesvesszőt nyomtat. Ennek megfelelően a ","-t valami másra, például ";"-re kell cserélni. A (ii) példában az elválasztó jelet - ami normál körülmények között szóköz, - a "*" karakterre cseréljük, míg a "$" jelet "@"-ra. Ha lenne Ft karakterünk, akkor arra is cserélhetnénk.

Hibalehetőségek A kifejezés kiértékelése után egy legfeljebb 4 hosszúságú sztringet kell kapnunk. Ha nem, akkor az interpreter ?ILLEGAL QUANTITY ERROR hibajelzést küld.

RCLR

Rövidítés: rC *Token*: $CD(205)

Mód: mind parancs, mind program módban használható.

Az aritmetikai függvény az i. színtípushoz rendelt szín kódját adja vissza. Az i lehetséges értékei:

i értéke	**Színtípus**
0	háttér színe
1	írás színe
2	több szín#1
3	több szín#2
4	keret színe

Szintaxis: RCLR(<aritmetikai kifejezés>)

Az argumentum értékének a 0-4 intervallumba kell esnie

Példák:

(i)
```
PRINT RCLR(4): REM = 14
```
(ii)
```
1000 DATA FEKETE,FEHER,PIROS,ENCIAN
1010 DATA BIBOR,ZOLD,KEK,SARGA
1020 DATA NARANCSSARGA,BARNA,KEKESZOLD,ROZSASZIN
1030 DATA KEKESZOLD,VILAGOSKEK,SOTETKEK,VILAGOSZOLD
1040 RESTORE 1000
1050 FOR I=0 TO 15: READ SZ$(I): NEXT I
1060 PRINT "<CLEAR>"
1070 PRINT "HATTER SZINE:";SZ$(RCLR(0))
1080 PRINT "KURZOR SZINE:";SZ$(RCLR(1))
1090 PRINT "KERET SZINE: ";SZ$(RCLR(4))
1100 PRINT "TOBBSZIN#1:  ";SZ$(RCLR(2))
1110 PRINT "TOBBSZIN#2:  ";SZ$(RCLR(3))
1120 GETKEY A$: RETURN
```

Az (i) példa a képernyőre írja a keret színének kódját. A (ii) alatt megírt alprogram kiírja valamennyi színtípus színét.

Hibalehetőségek Az aritmetikai kifejezések kiértékelése közben számos hiba adódhat. Ha értéke nem esik a 0-4 intervallumba, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

RDOT

Rövidítés:rD *Token*:$D0(208)

Mód: mind parancs, mind program módban használható.

Az argumentum értékétől függően a grafikus kurzor színtípusával, illetve x és y koordinátájával tér vissza:

érték	**Jelentés**
0	a grafikus kurzor x koordinátája
1	a grafikus kurzor y koordinátája
2	a grafikus kurzor színtípusa

Szintaxis: RDOT(<aritmetikai kif.>)

Az argumentum értékének a 0-2 intervallumba kell esnie.

Példák:

(i)
```
10 GRAPHIC 1,1
20 LOCATE 160,100
30 PRINT RDOT(0),RDOT(1): REM =160,100
40 PRINT RDOT(2): REM = 0 A HATTER SZINTIPUSA
```

A fenti egyszerű program az RDOT hatását mutatja be. A LOCATE utasítás a kurzort a (160,100) pontra állítja, ennek megfelelően RDOT(0),RDOT(1) értéke ugyanez. A nagyfelbontású képernyőt a 10. sorban töröltük, és azóta semelyik pontját sem kapcsoltuk be. Ennek megfelelően minden pont színe háttérszínű (=0).

Hibalehetőségek A kifejezés kiértékelése közben számos hiba adódhat. Ha értéke nem esik a 0-2 intervallumba, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

READ

Rövidítés: rE *Token*: $87(135)

Mód: mind parancs, mind program módban használható.

A program soraiban elhelyezett DATA utasításokban definiált értékeket olvassa és rendeli hozzá az utasításban megadott változókhoz.

Szintaxis: READ <változólista>

A <változólista> tetszőleges típusú, egymástól vesszővel elválasztott egyszerű- vagy tömbváltozókat tartalmaz. Legalább egy változó megadása kötelező.

Példák:

(i)
```
10 DATA 5,23,45,87,23,98
20 READ N
30 FOR I=1 TO N: READ X1%(N):NEXT I
```

(ii)
```
110 READ CIM:IF CIM<0 THEN LO%=(CIM+U) AND 255:
        HI%=(CIM+U)/256
```

(iii)
```
FOR I=1 TO 10:READ A:PRINT A,:NEXT
```

Az első példa azt szemlélteti, hogyan lehet a DATA utasítások első elemeként megadni a tárolt értékek számát. A READ utasítás segítségével először beolvassuk N értékét, majd egy 1..N ciklusban az adatokat.

A (ii) példa egy áthelyezhető gépi kódú program töltését mutatja. Amennyiben az érték negatív, a címet U hozzáadásával kell előállítani.

(iii) a READ parancs módban való használatát mutatja; a parancs a tárolt program következő 10 adatát olvassa be és írja ki a képernyőre.

Hibalehetőségek Ha a változó és a DATA utasításban talált érték típusa nem felel meg egymásnak, akkor ?TYPE MISMATCH ERROR vagy ?SYNTAX ERROR hibajelzést kapunk. Ha a DATA-ban levő adatok elfogytak, ?OUT OF DATA ERROR hibajelzést kapunk.

REM

Rövidítés: nincs *Token:*$8F(143)

Mód: mind parancs, mind program módban használható.

Lehetővé teszi megjegyzések beírását a program szövegébe. Az interpreter a REM észlelésétől a sor maradék részét már nem hajtja végre, hanem a **következő sor** elejéről folytatja a végrehajtást.

Szintaxis: REM <karaktersorozat>

A <karaktersorozat> tatszőleges karaktereket tartalmazhat; beleértve a kettőspontot is (:). A REM után - a következő sorig - bármi állhat.

Példák:

```
(i)    7000 REM *** FOPROGRAM ***
       7010 GOSUB 7040: REM *** KEZDOERTEKEK
       7020 GOSUB 7200: REM *** SZAMITASOK
       7030 GOSUB 8150: END: REM *** EREDMENYEK
       7040 REM *** SZUBRUTINOK

(ii)   10 REM "

       *** FOPROGRAM ***

(iii) X$=A$+CHR$(13)+A$+B$:GOSUB 2234:REMUB 2367
```

Az (i) példa a REM leggyakoribb felhasználását mutatja, amikor az utasítások hatását írja le. A (ii) példában a 10. sorban a REM-mel kapcsolatos egyik legegyszerűbb trükköt mutatjuk be; a REM után két <SHIFT-RETURN> karaktert szúrtunk be, aminek hatására a listázáskor a megjegyzés új sorban jelenik már meg. Ennek billentyűzése a következőképpen lehetséges:

```
10REM "  *** FOPROGRAM *** <SHIFT-RETURN>
<CRSR FEL>, nyolcszor <CRSR JOBBRA> <CTRL RVSON>
<SHIFT-M> <SHIFT-M> <RETURN>
```

Listázáskor a sor a fenti alakban jelenik meg.

A (iii) példa a REM parancs módban való használatát mutatja be. A program egy sorát kilistáztuk, töröltük a sorszámot és a nem kívánt rész elé egy REM-ot helyeztünk el. A <RETURN> megnyomása után a parancs a REM-ig hajtódik végre. (Az eredeti sor egy GOSUB 2367 utasítást tartalmazott.)

A LIST parancs a REM sorokban talált tokeneket nem grafikus jelként, hanem eredeti alapszavuknak megfelelően listázza ki.

Hibalehetőség Nincs.

RENAME

Rövidítés: reN *Token*: $F5(245)

Mód: mind parancs -, mind program módban használható.

A lemezen levő file átnevezésére használhatjuk. Paramétereiben a file régi és új nevét, továbbá a lemez meghajtójának adatait kell megadni.

Szintaxis:

```
                                               { ,  }
RENAME <régi név> TO<új név> [,D<arit.kif.>][ {    } U<arit.kif.> ]
                                               { ON }
```

A két aritmetikai kifejezés a lemezmeghajtó számát, illetve a lemezegység hardver számát adja meg. Ennek megfelelően az első értékének a 0-1, a második értékénke a 8-11 intervallumba kell esnie. Ha a paramétereket nem adjuk meg, akkor az utasítás a 8-as lemezegység 0-ás meghajtójában levő lemezre vonatkozik. Ha a <régi név>, illetve az <új név> paraméterek nem sztring konstansok, akkor kerek zárójelek közé kell zárni őket.

Példák:

```
(i)      RENAME "ZAJ" TO "ZOREJ"
(ii)     A$="ZAJ": B$="ZOREJ"
         RENAME (A$) TO (B$)
(iii)    RENAME "LOADER" TO "TOLTO" , D1 ON U9
```

Az (i) példa önmagáért beszél. A (ii) példa ugyanazt a feladatot látja el, mint az első, de bemutatja a paraméterek közt sztringváltozók használatát. A (iii) példa a 9-es lemezegység 1-es meghajtójában levő lemezen levő "LOADER" nevü file-t nevezi át "TOLTO"-re.

Hibalehetőségek Ha a paraméterek nem esnek a megengedett intervallumba, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk. Ha a szóbanforgó lemezegység nincs a rendszerben, akkor ?DEVICE NOT PRESENT ERROR hibaüzenetet kapunk. Ha a <régi név> nevü file nem létezik, akkor ?FILE NOT EXISTS ERROR hibajelzést kapunk. Ha a lemezre írás vagy olvasás közben hiba történik, akkor nem kapunk hibajelzést, csak a lemezegység lámpája villog. Ilyenkor a hibacsatorna kiolvasásával kapjuk meg a hiba pontos okát.

RENUMBER

Rövidítés: renU *Token*:$F8(248)

Mód: csak parancs módban használható.

A memóriában tárolt program sorait számozza át. A GOTO, GOSUB, IF, RESUME, TRAP, RESTORE utasításokban levő sorszámokat is átírja.

Szintaxis:
RENUMBER [<arit.kif.1>] [,<arit.kif.2>] [,<arit.kif.3>]

Az első aritmetikai kifejezés az átszámozott programrész első utasításának leendő sorszámát adja meg. A második, hogy az átszámozott programrész sorszámai mennyivel növekedjenek. Ha az első két paraméter valamelyike hiányzik, az interpreter a 10 értékkel dolgozik. A harmadik a memóriában levő program azon sorának a száma, ahonnan az átsorszámozást meg kell kezdeni. Ha a paraméter hiányzik, az interpreter 0-nak tekinti, azaz a teljes programot átszámozza.

Példák:
(i) RENUMBER
(ii) RENUMBER ,,4500
(iii) RENUMBER 0,1

Az (i) alatti parancsot használjuk talán a leggyakrabban. Hatására az átszámozott program sorai 10-zel kezdődnek és 10-zel növekednek. A (ii) parancsot általában akkor használjuk, ha egy meglevő programhoz - pl. a 4500 sortól kezdve - hozzáírtunk egy új részt. Ilyenkor csak a 4500., illetve az ennél nagyobb sorszámú sorokat számozza át a parancs. *Változások* a kisebb sorszámú sorokban is lesznek, hiszen a teljes programban átírásra kerülnek a módosuló hivatkozások.

Hibalehetőségek Ha a kezdő sorszám kisebb, mint a nem átszámozandó rész utolsó programsora, akkor az átszámozás ugyan megtörténik, de a sorok többé nem szerkeszthetők, s futás közben sem lehet rájuk hivatkozni.

RESTORE

Rövidítés: reS *Token*:$8C(140)

Mód: mind parancs, mind program módban használható.

Az utasítás hatására a READ utasítások a RESTORE utasításban megadott számú sortól kezdik el olvasni a DATA-kban tárolt adatokat. Ha a paraméter elmarad, az egy RESTORE 0 utasításnak felel meg.

Szintaxis: RESTORE [<sorszám>]

A <sorszám> csak előjel nélküli egész szám lehet.

Példák:

(i)
```
950 DATA ....
980 DATA ....
1000 DATA 169,0,141,....: REM GEPI KODU PROGRAM
1020 DATA ....
1340 RESTORE 1000
1345 :
1350 FOR L=828 TO 912:READ X:POKE L,X:NEXT
```

(ii)
```
2010 READ X$: IF X$="END" THEN RESTORE
```

Az első példa a <sorszám> használatát mutatja; hogyan lehet segítségével a DATA utasítások közt egy adott részt megtalálni. Először végrehajtjuk a RESTORE 1000 utasítást, majd ezt követően olvassuk be a gépi kódú rutin byte-jait.

A második példában az utolsó DATA tétel egy "END" sztring. Ennek az olvasása után a RESTORE a DATA mutatót az első utasításra állítja. (Nincs egy külön változó, amelyik azt mutatná, hogy a DATA-k elfogytak.)

A NEW, RUN, CLR utasítások egyben egy RESTORE-t is végrehajtanak. Program indítása előtt - parancs módban - a RESTORE-t így csak GOTO <sorszám> típusú indítás esetén érdemes használni.

Hibalehetőség Nincs. Előfordulhat azonban, hogy egy nem megfelelő sorszámú RESTORE miatt egy READ utasítás végrehajtása közben kapunk hibajelzést.

RESUME

Rövidítés: resU *Token:*$D6(214)

Mód: csak program módban használható.

A hibakezelő (TRAP) rutinból való kilépés jelzésére szolgál.

Szintaxis:

```
                  ⎧<sorszám>⎫
         RESUME [ ⎨         ⎬ ]
                  ⎩NEXT     ⎭
```

A <sorszám> előjel nélküli egész szám kell hogy legyen.

Példák:

(i)

```
10 TRAP 1000
20

....

1000 REM HIBAKEZELŐ RUTIN
1010 ?"A HIBAT FIGYELMEN KIVUL HAGYTUK
1020 RESUME NEXT
```

A példában egy olyan hibakezelő rutint mutatunk be, amelyik bármilyen hiba esetén kiírja a képernyőre az 1010. sorban látható szöveget, majd mintha nem is történt volna hiba, folytatja a program futását. Ezt a RESUME-ban használt NEXT paraméter segítségével érjük el.

Hibalehetőségek Ha a RESUME-ban megadott sor nem létezik, akkor a program futása ?UNDEF'D STATEMENT ERRROR hibajelzéssel megszakad.

RETURN

Rövidítés: reT *Token*:$8E(142)

Mód: mind parancs, mind program módban használható.

Az utasítás hatására a program vezérlése az utolsó GOSUB utasítást követő első utasításra kerül.

Szintaxis: RETURN

Példák:

(i)
```
10 INPUT X : GOSUB 1500 : GOTO 10
1500 RQ=0.5: X=INT((X-.005)/RQ+1)*RQ
1510 PRINT X;: RETURN
```

Egyetlen példánk az 1500. sorban kezdődő kerekítési eljárás ellenőrzését mutatja be. A 10. sorban különböző értékeket adhatunk a programnak és kerekített alakját ellenőrízhetjük.

Az utasítás szintaxisának ellenőrzése után a rutin végigolvassa a vermet FOR, DO és GOSUB tokeneket keresve. A FOR vagy DO tokeneket és a hozzájuk tartozó információt törli a veremből. A GOSUB megtalálása után visszatölti a megfelelő értékeket, majd megkeresi a következő kettőspontot (:) vagy a sor végét, és onnan folytatja a futást. Ha például az

```
ON L GOSUB 10,20,30: PRINT X
```

utasítás segítségével hajtottunk végre egy szubrutinhívást, akkor a veremből való visszatöltés után még a PRINT X is végrehajtódik.

Hibalehetőségek Ha az eljárás nem találja meg a visszatérési címet, ?RETURN WITHOUT GOSUB hibajelzést kapunk. A RETURN végrehajtása törli a verem tetején levő FOR-ra vonatkozó információkat. A

```
10 GOSUB 1000:NEXT J
1000 FOR J=0 TO 10:RETURN
```

program végrehajtása közben ezért ?NEXT WITHOUT FOR hibajelzést kapunk.

RGR

Rövidítés: rG *Token:*$CC(204)

Mód: mind parancs, mind program módban használható.

Az egyváltozós függvény a kiválasztott grafikus módnak megfelelő értékkel tér vissza. A változó álargumentum (dummy variable) csak a szintaxis miatt szerepel.

érték	Kijelzési mód
0	normál, karakteres képernyő
1	normál, nagyfelbontású képernyő
2	normál, vágott képernyő
3	többszínű, nagyfelbontású képernyő
4	többszínű, vágott képernyő

Szintaxis: RGR(<kifejezés>)

Példák:

```
(i)     GRAPHIC 1,1: ? RGR(0): REM = 1
        GRAPHIC 3: ? RGR(121): REM = 3
        GRAPHIC 3: ? RGR(X+Y): REM = 3
```

Példáink az RGR hatását mutatják be. Az utolsó két sor mutatja, hogy az RGR függvény értéke független az argumentumától.

Hibalehetőségek Nincs.

RIGHT$

Rövidítés: rI (a $ jel már nem kell!) *Token:* $C9(201)

Mód: mind parancs, mind program módban használható.

A sztring függvény az argumentum sztring **jobb oldali** karaktereiből egy új sztringet képez. A felhasználandó karakterek **számát** a függvény második paramétere határozza meg.

Szintaxis: RIGHT$(<sztring kifejezés>, <aritmetikai kifejezés>)

Az aritmetikai kifejezés értéke nem lehet 255-nél nagyobb.

```
Legyen     X$="KOVACS PETER"
    Pozició= 123456789012

Ekkor RIGHT$(X$,5)="PETER"
      RIGHT$(X$,25)=X$.
```

Példák:

```
(i)   10 PRINT RIGHT$("ABRAKDABRA",3):REM=BRA
(ii)  20 PRINT RIGHT$("          "+STR$(N),10)
```

Az első példa a RIGHT$ hatását mutatja be, a második az N értékének megfelelő számot jobbra tömörítve, 10 karakterpozíción írja ki.

A RIGHT$ függvény helyettesíthető a MID$ függvénnyel:

RIGHT$(X)=MID$(X$,LEN(X$)-N+1)

Hibalehetőségek A paraméter értéke nem eshet a megadott érték-határokon kívül.

RLUM

Rövidítés: rL *Token*: $CE(206)

Mód: mind parancs, mind program módban használható.

Az argumentumként megadott színtípus fényerejének értékét adja meg. Az egyes színtípusok jelentése a következő:

érték	**Színtípus**
0	háttér színe
1	írás színe
2	több szín#1
3	több szín#2
4	keret színe

Szintaxis: RLUM(<arit.kif>)

Az aritmetikai kifejezés értékének a 0-4 intervallumba kell esnie.

Példák:

```
(i)    PRINT RLUM(4): REM = 6
(ii)   FOR I=0 TO 5: ? RLUM(I): NEXT I
```

Az (i) alatti parancs a keret színének a fényerejét írja ki. A (ii) alatti parancs valamennyi színtípus fényerejének értékét kiírja.

Hibalehetőségek Ha az argumentum értéke nem esik a 0-4 intervallumba, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

RND

Rövidítés: nincs *Token:*$BB(187)

Mód: mind parancs, mind program módban használható.

A 0-1 zárt intervallumba eső **pszeudo-véletlen számokat generál.**

Szintaxis: RND(<aritmetikai kifejezés>)

Az aritmetikai kifejezés értékének csak az **előjele** számít.

Példák:

```
(i)   10 FOR J=0 TO 3000*RND(1):NEXT
(ii)  700 DATA 11,22,33,44,55,66,77,88,99
      710 FOR J=1 TO 9*RND(1):READ X$: NEXT J
(iii) 200 X=(B-A)*RND(1)+A
```

Az első példa a program futását maximum három másodpercre - de véletlen hosszúságú ideig - felfüggeszti. A második példa a kilenc elemü DATA listát véletlenszerű eleméig olvassa. Igy tudunk diszkrét értékek közül véletlenszerűen választani.

Utolsó, (iii)-as példánk az [A,B] intervallumba eső véletlen számot állít elő.

Az RND utasítás természetesen nem állít elő 'igazi' véletlen számokat. Az argumentum előjele határozza meg, hogy milyen eljárás segítségével számítja ki az interpreter a következő számot. Az interpreter az utoljára előállított véletlen számot külön tárolja, és bizonyos számelméleti függvények segítségével kapja meg a következőt. Amennyiben az RND függvény argumentuma pozitív, az interpreter az így képzett véletlen számot generálja. **Negatív** argumentum az **első véletlen számot** állítja **elő**; X=RND(-1):PRINT X **értéke mindig ugyanaz.** Végül RND(0) a VIA chipek óra-regisztereit használja fel véletlen számok előállítására. Ez 'inkább' véletlen lesz, mintha csak a számelméleti függvényeket használnánk, de ez sem teljesen véletlen. Ha például egy cikluson belül ismételten használjuk a RND(0) utasítást, akkor bizonyos szabályosság azonnal jelentkezik a végrehajtási időben.

Hibalehetőségek Az aritmetika kifejezés kiértékelése okozhat csak hibát.

RUN

Rövidítés: rU *Token:*$8A(138)

Mód: mind parancs, mind program módban használható.

A memóriában tárolt programot az elejéről, vagy egy adott ponttól kezdve végrehajtja. A változók előző értéke elvész, a RUN ugyanis végrehajt egy CLR és egy RESTORE utasítást is.

Szintaxis: RUN [<sorszám>]

Példák:

```
(i)    RUN
(ii)   RUN 1000
(iii)  5000 IF X<>0 THEN RUN
```

Az első két sor a RUN parancs módban való használatát szemlélteti. Az utolsó példában, ha X=0, akkor a RUN utasítás törli az összes változót és a program elölről kezd el futni. Ha a RUN utasítást nem követi egy kettőspont (:), akkor az interpreter sorszámot tételez fel. Igy RUN X és RUN "PRG" hatása egyaránt RUN 0 (lásd a GOTO utasítást!). Az X=80: RUN X hatására a program nem a 80., hanem a 0. sortól kezd futni, de nem kapunk hibajelzést!

A <SHIFT-RUN> billentyű lenyomása ekvivalens a

```
DLOAD "*"<RETURN>
RUN  <RETURN>
```

billentyűzésével, tehát a lemezről betölti az első programot, és azonnal el is indítja.

Hibalehetőség Ha a RUN utasításban megadott sorszámú programsor nem létezik, akkor ?UNDEF'D STATEMENT ERROR hibajelzést kapunk.

SAVE

Rövidítés: sA *Token:*$94(148)

Mód: mind parancs, mind program módban használható.

Az utasításban megadott nevű file-ba **átmásolja (elmenti)** a memóriában tárolt BASIC programot.

Szintaxis:
SAVE[<sztring kif.>] [,<arit. kif.>] [,<arit. kif.>]

A második <aritmetikai kifejezés> egyedül a kazettás egység esetén érdekes. Ha értéke 2, akkor a file elmentése után még egy EOT jel is kiíródik a szalagra. (Lásd a kazettás egységről szóló fejezetet.)

Példák:

```
(i)   SAVE      : REM NEV NELKUL A KAZETTAN
(ii)  SAVE "",8: REM HIBAJELZEST KAPUNK, "" NEM NEV
(iii) SAVE "PROG",8:REM A PROGRAM NEVE PROG=5 LESZ A LEMEZEN
(iv)  SAVE "@:REGI",8:REM LEMEZEN MAR MEGLEVO FILEBA
(v)   12 SAVE "TEST"+TI$,8
```

A fenti példák - ugy hisszük - önmagukért beszélnek. Az (i) példa a kazettás egységen egy **név nélküli** file-t hoz létre. Ilyen a lemezegységen nem lehet, ezért a (ii) példa hibás.

A (iii) példa a lemezegység leggyakoribb használatát mutatja. A memóriában tárolt program kiíródik a lemezre egy "PROG" nevű file-ba. Ha egy ilyen nevű file már létezett, akkor az elmentés nem történik meg. A (iv) példa mutatja, hogyan érhetjük el, hogy a már meglevő file-t felülírjuk. A (iv) alatti parancs kiadása mindig elmenti a tárolt programot. Ha a lemezen volt már egy "REGI" nevű program, annak az előző tartalma elvész, és helyére a jelenleg a memóriában levő program kerül.

Az utolsó példa a SAVE utasítás programból való használatát mutatja; a programrész időről időre a tárolt programot kiírja a memóriából. Hogy ezek a file-ok különbözőek legyenek, a név utolsó hat karakterét a TI$ adja, ami mindig más és más. (A dolognak csak akkor van persze értelme, ha a program folyamatosan módosítja önmagát...)

Hibalehetőségek Ha az utasításban megjelölt egység nincs bekapcsolva, vagy hibásan működik, akkor ?DEVICE NOT PRESENT hibaüzenetet kapunk. A kazettás egységre való kimentés aszinkron. Ha például csak a <PLAY> billentyűt nyomjuk meg, az utasítás hibajelzés nélkül végrehajtódik, bár a szalagra a program természetesen nem kerül kiírásra. Ha a lemezen már

SAVE

Rövidítés: sA *Token:*$94(148)

Mód: mind parancs, mind program módban használható.

Az utasításban megadott nevű file-ba **átmásolja (elmenti)** a memóriában tárolt BASIC programot.

Szintaxis:
SAVE[<sztring kif.>] [,<arit. kif.>] [,<arit. kif.>]

A második <aritmetikai kifejezés> egyedül a kazettás egység esetén érdekes. Ha értéke 2, akkor a file elmentése után még egy EOT jel is kiíródik a szalagra. (Lásd a kazettás egységről szóló fejezetet.)

Példák:

```
(i)   SAVE      : REM NEV NELKUL A KAZETTAN
(ii)  SAVE "",8: REM HIBAJELZEST KAPUNK, "" NEM NEV
(iii) SAVE "PROG",8:REM A PROGRAM NEVE PROG=5 LESZ A LEMEZEN
(iv)  SAVE "@:REGI",8:REM LEMEZEN MAR MEGLEVO FILEBA
(v)   12 SAVE "TEST"+TI$,8
```

A fenti példák - ugy hisszük - önmagukért beszélnek. Az (i) példa a kazettás egységen egy **név nélküli** file-t hoz létre. Ilyen a lemezegységen nem lehet, ezért a (ii) példa hibás.

A (iii) példa a lemezegység leggyakoribb használatát mutatja. A memóriában tárolt program kiíródik a lemezre egy "PROG" nevű file-ba. Ha egy ilyen nevű file már létezett, akkor az elmentés nem történik meg. A (iv) példa mutatja, hogyan érhetjük el, hogy a már meglevő file-t felülírjuk. A (iv) alatti parancs kiadása **mindig** elmenti a tárolt programot. Ha a lemezen volt már egy "REGI" nevű program, annak az előző tartalma elvész, és helyére a jelenleg a memóriában levő program kerül.

Az utolsó példa a SAVE utasítás programból való használatát mutatja; a programrész időről időre a tárolt programot kiírja a memóriából. Hogy ezek a file-ok különbözőek legyenek, a név utolsó hat karakterét a TI$ adja, ami mindig más és más. (A dolognak csak akkor van persze értelme, ha a program folyamatosan módosítja önmagát...)

Hibalehetőségek Ha az utasításban megjelölt egység nincs bekapcsolva, vagy hibásan működik, akkor ?DEVICE NOT PRESENT hibaüzenetet kapunk. A kazettás egységre való kimentés aszinkron. Ha például csak a <PLAY> billentyűt nyomjuk meg, az utasítás hibajelzés nélkül végrehajtódik, bár a szalagra a program természetesen nem kerül kiírásra. Ha a lemezen már

SCNCLR

Rövidítés: sC   *Token:* $E8(232)

Mód: mind parancs, mind program módban használható.

A kiválasztott grafikus módnak megfelelő képernyő(ke)t törli.

Szintaxis: SCNCLR

Példák:

```
(i)     GRAPHIC 1: SCNCLR
(ii)    GRAPHIC 3: SCNCLR
```

Az (i) példa törli a nagyfelbontású képernyőt. A (ii) példában vágott képernyőt használunk, ennek megfelelően a SCNCLR törli a ***teljes*** karakteres és a ***teljes*** nagyfelbontású képernyőt.

Hibalehetőségek Nincs. (De a képernyő tartalma elvész!)

SCRATCH

Rövidítés: sC   *Token:* $F2(242)

Mód: mind parancs, mind program módban használható.

A paramétereiben megadott meghajtóban levő, adott nevű file-t törli a lemezről.

Szintaxis:

```
                                  ⎧ ,  ⎫
SCRATCH <név> [,D<arit.kif.1>] [  ⎨    ⎬U<arit.kif.2>  ]
                                  ⎩ ON ⎭
```

A két aritmetikai kifejezés a lemezegység meghajtóját, illetve a hardver számát adja meg. Ha a név nem sztring konstans, akkor kerek zárójelek közé kell tenni. A <név>-ben a * és ? karakterek jelentése a szokásos.

Példák:

```
(ia)    SCRATCH "KUKAC"
(ib)    X$="KUKAC": SCRATCH (X$)
(ii)    SCRATCH "TEMP*",D1
(iii)   SCRATCH "????.PRG",DO ON U10
```

Az (ia) és (ib) alatti parancsok egyaránt a 8-as lemezegység 0-ás meghajtójában levő lemezről törlik a "KUKAC" nevü file-t. A

(ii) parancs törli a 8-as lemezegység 1-es meghajtójában levő lemezről az összes "TEMP"-pel kezdődő nevű file-t.

A (iii) parancs a 10-es lemezegység 0-ás meghajtójában levő összes olyan file-t törli, amelyik neve pontosan 8 karakterből áll, s az utolsó négy karakter ".PRG"

A parancsok végrehajtása előtt a rendszer visszakérdez:

ARE YOU SURE? (Biztos benne?)

Ha igen, akkor a <Y> billentyűt kell válaszként megnyomni. Erre az interpreter elküldi a lemezegységnek a megfelelő parancsot. Ha az <N> billentyűt nyomjuk meg, a parancs figyelmen kivül marad.

Hibalehetőségek A kifejezések kiértékelése közben számos hiba adódhat. Ha az aritmetikai kifejezések értéke nem megfelelő akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk. Ha a lemezegység nincs a rendszerben, akkor ?DEVICE NOT PRESENT ERROR hibajelzést kapunk. Ha a törlés közben írási vagy olvasási hiba történik, akkor erről csak a hibacsatorna kiolvasásával szerezhetünk információt.

SGN

Rövidítés: sG *Token*:$B4(180)

Mód: mind parancs, mind program módban használható.

Az aritmetikai függvény az argumentum előjelétől függően a következőt számítja ki:

$$SGN(X)=\begin{cases} +1 & \text{ha } X>0; \\ 0 & \text{ha } X=0; \\ -1 & \text{ha } X<0. \end{cases}$$

Szintaxis: SGN(<aritmetikai kifejezés>).

Példák:
```
(i)   10 IF SGN(X)>0 THEN PRINT X;"POZITIV"
(ii)  20 IF ABS (SGN(X))<>0 THEN PRINT X;"NEM NULLA"
(iii) 30 ON SGN(X)+2 GOSUB 100,110,120
```

Az első két sor az SGN egyszerű használatát mutatja. A harmadik példa a FORTRAN aritmetikai GOTO utasitását szimulálja. (FORTRAN ekvivalens: IF(X) 100,200,300). A vezérlés a 100., 110. illetve 120. sorba kerül, attól függően, hogy X negatív, nulla vagy

pozitív volt-e.

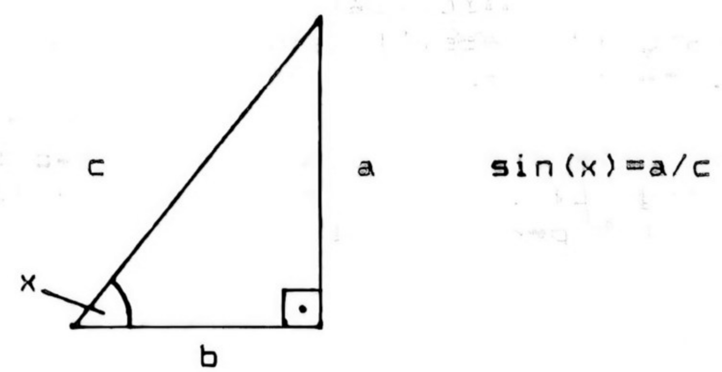
Hibalehetőségek Csak az argumentum kiértékelése okozhat hibát.

SIN

Rövidítés: sI *Token:* $BF(191)

Mód: mind parancs, mind program módban használható.

A radiánban megadott szög szinuszát számítja ki. Az alábbi ábra illusztrálja x és sin(x) viszonyát:



Szintaxis: SIN(<aritmetikai kifejezés>)

Példák:

```
(i)    10 PRINT SIN(1)        : REM =.841470985
(ii)   20 PRINTSIN(360*<pi>/180): REM =0
(iii)  100 PRINT"<CLR>":FOR I=1 TO 24
       110 PRINT TAB(SIN(I/3)*19+20);"*":NEXT I
(iv)   200 X=A+SIN(A):Y=A+SIN(A/2)*2
```

Az első két példa önmagáért beszél. A harmadik program egy szinuszgörbét rajzol a képernyőre. A negyedik példa egy rajzoló program része.

Az argumentum nagysága a végeredmény pontosságát nem befolyásolja; a számítás során az interpreter ugyanis elosztja 2*pi-vel az argumentumot, és a maradékkal számol tovább.

Hibalehetőségek Az aritmetikai kifejezés kiértékelése közben számos hibalehetőség adódik. Ha ez hiba nélkül befejeződött, a SIN rutin már hibátlanul lefut.

SOUND

Rövidítés: sO *Token*:$DA(218)

Mód: mind parancs, mind program módban használható.

A hanggenerátor chip meghatározott oszcillátorát adott ideig és adott hangmagassággal bekapcsolja.

Szintaxis:SOUND (<arit.kif.1>,<arit.kif.2>,<arit.kif.3>)

Az első aritmetikai kifejezés a kiválasztott oszcillátor száma:

érték	**oszcillátor**
0	zenei hang (1. oszcillátor)
1	zenei hang (2. oszcillátor)
2	zaj (3. oszcillátor)

A 2. és 3. aritmetikai kifejezés a hang magasságát, illetve kitartási idejét adja meg. Ez utóbbi legnagyobb értéke 65535, ami 1311 másodpercnek (21.8 perc) felel meg.

Példák:
```
(i)    SOUND 2,800,6000
(ii)   SOUND 1,770,300
```

Az (i) példa két percig a 800 értéknek megfelelő frekvenciájú ún. *fehér zajt* produkál. (ii) rövid időre megszólaltatja a normál zenei 'A' hangot.

A hangmagasságot megadó kifejezés értékének, M-nak a 0-1023 intervallumba kell esnie. Az M értékének megfelelő hang

$$H = \frac{111840.45}{M-1024}$$

magasságú lesz (Hz-ben számolva.)

Hibalehetőségek Az aritmetikai kifejezések értékének a jelzett intervallumba kell esnie. Ha nem, akkor ?ILLEGAL QUANTITY ERRROR hibaüzenetet kapunk.

SPC(

Rövidítés: sP (beleértve a kezdő zárójelet) *Token:*$A6(166)

Mód: mind parancs, mind program módban használható.

Adott számú szóközt, vagy <CRSR JOBBRA> karaktert nyomtat.

Szintaxis: SPC(<aritmetikai kifejezés>)

Az utasítás csak a PRINT utasításban szerepelhet. Az SPC és a (jel közt nem lehet szóköz. Az <aritmetikai kifejezés> értékének - lefelé kerekítés után - a 0-255 intervallumba kell esnie.

```
10 PRINT" ": FOR J=0 TO 20: PRINT"*";SPC(20)"*";:NEXT
20 FOR J=1 TO 19:PRINT SPC(J)"*"SPC(20-J*2)"*":NEXT
```

A fenti példák azt illusztrálják, hogy egy ciklusban használt SPC(utasítás segítségével hogyan rajzolhatunk a képernyőre. Az első program egy keretet rajzol, a második pedig egy V alakú alakzatot.

Az, hogy a SPC szóközt vagy <CRSR JOBBRA> karaktert nyomtat, a programból állitható. Ha a 19.($13) címen 0 található, akkor a SPC a kurzort mozgatja, ha ettől eltérő az érték, akkor szóközöket ír ki, ahogy ezt az alábbi program illusztrálja:

```
10 INPUT A
20 POKE 19,A
30 PRINT "<CLR>";:FOR J=0 TO 21: PRINT "X";:NEXT
40 PRINT "<HOME>**"TAB(7)"**"SPC(5)"**"
50 POKE 19,0
```

Az eredmény:

```
**     **     **XXXXX (A=1 esetén)
**XXXXX**XXXXX**XXXXX (A=0 esetén)
```

Hibalehetőségek Ha az aritmetikai kifejezés értéke nem esik a 0-255 intervallumba, ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

SQR

Rövidítés: sQ *Token:* $BA(186)

Mód: mind parancs, mind program módban használható.

Az argumentum négyzetgyökét számítja ki.

Szintaxis: SQR(<aritmetikai kifejezés>)

Példák:

```
(i)   PRINT SQR(2): REM = 1.4142...
(ii)  PRINT SQR(9)^2: REM = 9
(iii) X1=(-B+SQR(B*B-4*A*C))/(2*A)
      X2=(-B-SQR(B*B-4*A*C))/(2*A)
(iv)  2000 T=SQR(X*X+Y*Y+Z*Z)
```

A SQR függvényre - éppen úgy, mint az EXP-re - valójában nincs szükség, hiszen SQR(X)=X^.5. Az első két példa az SQR hatását mutatja be. A következő két sorban az A,B,C együtthatójú másodfokú egyenlet megoldásait számítottuk ki. Utolsó példánk az (X,Y,Z) pontnak az origótól vett távolságát adja.

Hibalehetőségek Ha az argumentum értéke negatív, ?ILLEGAL QUANTITY ERROR hibajelzést kapunk. Vigyázzunk, sok programnyelvben SQR - éppen ellenkezőleg, mint a C-16 esetében - a szám négyzetét számítja ki!

SSHAPE

Rövidítés: sS *Token:* $E4(228)

Mód: mind parancs, mind program módban használható.

A nagyfelbontású képernyő egy téglalap alakú részét - a GSHAPE utasítás segítségével visszatölthető alakban - átmásolja egy sztringváltozóba.

Szintaxis: SSHAPE <sztring vált.>,<x1>,<y1> [,<x2>,<y2>]

Az <x1>,<y1> az átmásolandó téglalap bal felső, míg <x2>,<y2> a jobb alsó csúcsának koordinátái. Ha ez utóbbi elmarad, akkor a grafikus kurzor helyzete határozza meg a bal alsó sarkot. <x1>,<y1>,<x2>,<y2> tetszőleges aritmetikai kifejezések lehetnek. A <sztring vált.> hossza normál, illetve több színű üzemmódban a következőképpen alakul:

```
INT((ABS(X1-X2)+1)/4+.99)*(ABS(Y1-Y2)+1)+4      (Normál)
INT((ABS(X1-X2)+1)/8+.99)*(ABS(Y1-Y2)+1)+4      (Több szinű)
```

Példák: Lásd a GSHAPE-nél szereplő példát.

Hibalehetőségek A kifejezések kiértékelése közben számos hiba adódhat. Ha a téglalap koordinátáit úgy adjuk meg, hogy a sztring hossza nagyobb lenne, mint 255, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

ST

Fenntartott BASIC változó.

ST értéke a perifériális egységek mindenkori állapotáról tájékoztat. ST jelentése az egység típusától függően más és más. A következő táblázat ST lehetséges értékeit és azok jelentését foglalja össze:

Bit	**érték**	**Kazettás egység I/O**	**Soros busz**	**Kazettás egység Verify+Load**
0	1		idő túllépés	
1	2		idő túllépés	
2	4	rövid blokk		rövid blokk
3	8	hosszú blokk		hosszú blokk
4	16	hiba olvasási hiba		
5	32	ellenőrző összeg hiba		ellenőrző összeg hiba
6	64	file vége jel	file vége jel	EOT
7	-128		nincs bekapcsolva	

Az ST értéke a GET, INPUT és a PRINT, továbbá a CMD, GET#, INPUT# és a PRINT# utasitások megkezdése előtt nulla lesz. Ha a korábbi értékére a későbbiekben szükség lesz, akkor ST értékét egy másik változóba kell tárolni. Az ST-ben tárolt információt **valamennyi** I/O utasítás után ellenőrizni lehet.

ST nullától különböző értéke nem feltétlenül jelent hibát (pl. ST=64). Az ST értékét az interpreter egy byte-os számként a 144($90) címen tárolja.

STOP

Rövidítés: sT *Token*:$90(144)

Mód: mind parancs, mind program módban használható.

Megállitja a program futását és kiirja, hogy hányadik sorban állt meg. A program futása a CONT paranccsal folytatható.

Példák

A STOP utasítást elsősorban a programfejlesztés stádiumában használjuk, hiszen kész programok esetén érdektelen, hogyan dolgozik a program, és hol is állt meg. A STOP utasítás segítségével töréspontokat helyezhetünk el a programban:

```
(i)  10 GOSUB 2000: STOP: GOSUB 3000: STOP
(ii) 1254 IF G$="" THEN STOP
```

A STOP végrehajtása után a program változóit a PRINT utasítás segítségével ellenőrizhetjük. Ezután a CONT még müködik. Ha egy régi vagy új változónak adunk értéket a CONT általában még mindig használható. Ha újraszerkesztjük a programot, a CONT már nem használható, és a változók értéke is elveszett.

Hibalehetőségek Nincs.

STR$

Rövidítés: stR (beleértve a $-t is) *Token*:$C4(196)

Mód: mind parancs, mind program módban használható.

Az argumentumként megadott aritmetikai kifejezés értékének sztring alakját adja meg.

Szintaxis: STR$(<aritmetikai kifejezés>)
A STR és a $ jel közt nem lehet szóköz. Az <aritmetikai kifejezés> értéke a PRINT formátumának megfelelő alakban áll elő. Igy például STR$(.005) nem ".005", hanem " 5E-03" lesz!

Példák:

```
(i)   PRINT STR$("0.0024"): REM =" 2.4E-03
(ii)  10 PRINT STR$(555)+".00": REM = 555.00
(iii) 20 N$="0"+MID$(STR$(N),2): PRINT N$
```

Az utasítást elsősorban a számok alakjának szerkesztésére használjuk. Az első két példa egyszerűen csak szemlélteti a STR$

hatását. A (iii) példa az 1-nél kisebb számokat elõnullázva írja ki. Például .05-t a program 0.05 alakban írja ki.

Hibalehetõség A számok normális írása és a STR$(N) alak nem mindig egyezik meg. Ebbõl szemantikus hibák adódhatnak. Az argumentum kiértékelése további hibák forrása lehet.

SYS

Rövidítés: sY *Token:* $9E(158)

Mód: mind parancs, mind program módban használható.

A vezérlés a SYS argumentumán kezdõdõ **gépi-kódú alprogramra** adódik át. A RTS gépi kódú utasítás végrehajtása után a vezérlés a BASIC-be kerül vissza.

Szintaxis: SYS <aritmetikai kifejezés>

Példák:
A SYS 64802 parancs kiadása ekvivalens a számítógép ki-, és bekapcsolásával.

Hibalehetõségek Az utasítás hatására a program futásának ellenõrzése kikerül a BASIC interpreter hatálya alól. Ha a gépi-kódú rutint nem jól írtuk meg, a C-16 teljesen 'lemerevedhet'. Ilyenkor nincs mást tenni, mint kikapcsolni a gépet. SYS-t tartalmazó program kipróbálása elõtt feltétlenül mentsük el a programot!

TAB(

Rövidítés: tA (beleértve a kezdő zárójelet is) *Token:* $A3(163)

Mód: mind parancs, mind program módban használható.

Amennyiben a kurzor pozíciója az adott sorban kevesebb, mint a TAB(paraméterében megadott érték, addig a pozícióig szóközöket, vagy <CRSR JOBBRA> karaktereket nyomtat.

Szintaxis: TAB(<aritmetikai kifejezés>)

Az utasitás csak a PRINT utasításban szerepelhet. A TAB és (jel közt nem lehet szóköz.

Példák:

```
(i)  10 PRINT TAB(40/2-LEN(N$)/2);N$
(ii) 20 PRINT"*";TAB(255);"*"
```

Az első példa a képernyő közepére tabulálja az N$-ban tárolt üzenetet. A következő példa azt szemlélteti, hogy a TAB(hatása több soron keresztül is érvényesülhet.

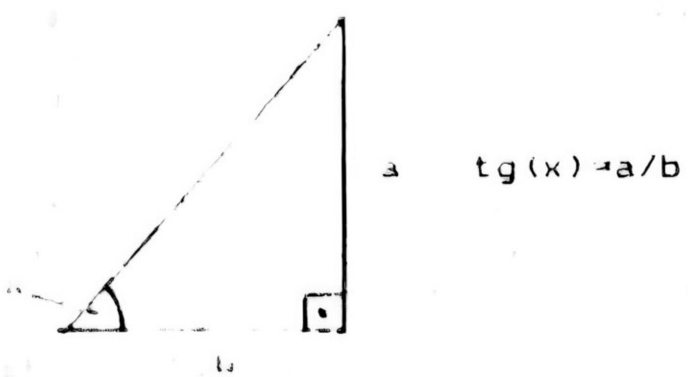
Hibalehetőség Ha az aritmetikai kifejezés értéke nem esik a 0-255 intervallumba, akkor ?ILLEGAL QUANTITY ERROR hibajelzést kapunk.

TAN

Rövidítés: nincs *Token:* $C0(192)

Mód: mind parancs, mind program módban használható.

A radiánban megadott argumentum tangensét számítja ki. Az alábbi ábra mutatja, hogyan lehet egy háromszög oldalaiból a tg(x) függvényt kiszámítani:



Szintaxis: TAN(<aritmetikai kifejezés>)

Példák:

```
(i)   PRINT TAN(<pi>/2): REM ?OVERFLOW ERROR
(ii)  PRINT TAN(45/57.29578):REM = TG 45 fok
(iii) 30 X=(TAN(A)+TAN(B))/(1-TAN(A)*TAN(B))
```

Az első két példa a TAN hatását mutatja be, a másodikban a fokokban adott szöget először átszámítjuk radiánba. A harmadik példa a tangens függvényre vonatkozó addiciós tételt használja; X valójában TAN(A+B)-vel egyenlő.

Hibalehetőség Amikor cos(x) közelítőleg nulla, akkor ?OVERFLOW ERROR hibaüzenetet kaphatunk. (tg(x)=sin(x)/cos(x) miatt.)

TI és TI$

Fenntartott BASIC változók.

TI és TI$ a C-16 belső órájának segítségével a bekapcsolás óta eltelt időt mérik. TI valós változó, ami azonban csak olvasható. A 0. lapon található 'szabadon futó óra' másodpercenként körülbelül 60-szor aktualizálódik, így TI/60 a gép bekapcsolása óta eltelt másodperceket mutatja. TI$ egy hat karakteres sztring, amelynek két-két karaktere rendre az órát, a percet és a másodpercet adja. TI$ értéke értékadó utasítással változtatható.

```
(i)   TI$="081500" : REM (ha 1/4 9-kor kezdtünk el dolgozni.
      ?TI$: REM mielött kikapcsoljuk a gépet.
(ii)  100 T=TI
      110 IF TI-T<120 THEN GOTO 110
(iii) 1000 PRINT MID$(TI$,1,2);":";MID$(TI$,3,2);":";
      1020 PRINT MID$(TI$,5,2)
(iv)  5000 TR=0
      5010 <PROGRAM>
           ....
      8020 PRINT (TI-TR)/60 : END
```

Az első példa azt mutatja, hogy TI$ értékét magunk is megadhatjuk. A sztringnek, ami TI$ új értékét definiálja, pontosan hat karakter hosszúnak kell lennie. A (ii) példa 2 másodpercnyi időre felfüggeszti a program futását (ciklusban várakozik).

A (iii) példa az időt a digitális órákon szokásos hh:mm:ss alakban jelzi ki. Utolsó példa egy tetszőleges program futási idejének kiszámítására szolgál. A program futása előtt a TI-ben

tárolt értéket a TR változóban elmenti, majd a program végét jelentő END végrehajtása előtt az azóta eltelt időt másodpercekben kiírja.

Hibalehetőség Ha olyan gépi-kódú rutint, vagy BASIC parancsot használunk, amelyik letiltja a megszakításokat, az 'óra' pontatlanná válik, késik.

TRAP

Rövidítés: tR *Token:*$D7(215)

Mód: csak program módban használható.

Saját hibakezelő rutin kezdőcímét adja meg. Ha az utasítást követően hiba történik, akkor a vezérlés a TRAP-ben megadott címre kerül. A hibakezelő rutinból a RESUME utasítás kiadásával léphetünk ki.

Szintaxis: TRAP [<sorszám>]

A <sorszám> csak előjel nélküli egész szám lehet. Ha hiányzik, az a 0 értéknek felel meg.

Példák:
(i) TRAP 1200

Az utasítás végrehajtását követően bármely hiba után a vezérlés az 1200-as sorra kerül. A program folytatása a hibakezelő rutin által elsőnek végrehajtott RESUME-nak megfelelően történik.

Hibalehetőség Ha a TRAP-ban szereplő sorszámú sor nem létezik, akkor nem kapunk hibajelzést. Ha ezek után történik hiba, akkor a program futása hibajelzés nélkül megszakad.

TROFF és TRON

Rövidítés: troF *Token:* TROFF $D9(217)
trO TRON $D8(216)

Mód: mind parancs, mind program módban használható.

A TRON bekapcsolja, a TROFF pedig kikapcsolja a nyomkövetési üzemmódot. Nyomkövetési üzemmódban a végrehajtott BASIC utasítások sorszámait az interpreter szögletes zárójelek közé téve folyamatosan kiírja a képernyőre. Ha programból használjuk, akkor lehetőség van arra, hogy csak a program egyes részeit nyomkövessük.

Szintaxis: TRON
TROFF

Hibalehetőségek Nincs.

UNTIL

Rövidítés: uN *Token:*$FC(252)

Mód: mind parancs, mind program módban használható.

A DO vagy LOOP utasításban levő kilépési feltétel megadására használhatjuk. A ciklus akkor jár le, ha az UNTIL utáni logikai kifejezés igazzá válik.

Szintaxis:

```
 ⎧DO  ⎫
[⎨    ⎬] UNTIL <logikai kifejezés>
 ⎩LOOP⎭
```

Példák

```
(i)   DO UNTIL JOY(1)>0: LOOP
(ii)  1200 DO
      1210 INPUT "<CLEAR>X=";X
      1220 LOOP UNTIL X=INT(X)
```

Az (i) parancsban levő ciklus addig nem fejeződik be, míg az UNTIL-t követő kifejezés igaz nem lesz, azaz egészen addig, amíg az 1-es botkormánnyal nem csinálunk valamit. (ii) egy beolvasó ciklus. A ciklus az 1210-es INPUT utasítást addig ismétli, míg X-nek egész értéket nem adunk.

Hibalehetőségek A logikai kifejezés kiértékelése közben számos hiba történhet. További hibát okozhat az UNTIL-t tartalmazó DO vagy LOOP nem megfelelő használata.

USR

Rövidítés: uS *Token:*$B7(183)

Mód: mind parancs, mind program módban használható.

Aritmetikai függvény, amelyik a felhasználó által definiált gépi-kódú programot hajtja végre.

Szintaxis: USR(<aritmetikai kifejezés>)
Az aritmetikai kifejezés értékének a 0-65535 intervallumba kell esnie (kerekítés után). Ezenkívül a 1280-1282 címen egy gépi-kódú utasításnak kell lennie (ez általában JMP ($xxxx) alakú).

Hibalehetőségek A BASIC interpreter nem ellenőrzi a gépi-kódú program futását. A USR használatára ugyanaz vonatkozik, mint a SYS-re.

VAL

Rövidítés: vA *Token:*$C5(197)

Mód: mind parancs, mind program módban használható.

Az argumentumként szereplő sztring kifejezés numerikus értékét számítja ki. A sztringet első - szám részeként már nem értelmezhető - karakteréig dolgozza fel.

Szintaxis: VAL(<sztring kifejezés>)

A <sztring kifejezés> értékeként egy legfeljebb 255 hosszú sztringet kell kapnunk.

Példák:
```
10 PRINT VAL("123.321")     : REM = 123.321
20 PRINT VAL("-123")        : REM = -123
30 PRINT VAL("1.2 E2")      : REM = 120
40 PRINT VAL("E")           : REM = 0
50 PRINT VAL("10000000000"): REM = 1E+10
60 PRINT VAL(LEFT$(TI$,2):  REM = 0-24 KOZT
```

```
70 PRINT VAL("123+321")     : REM = 123
80 PRINT VAL("1.2.3")       : REM = 1.2
90 PRINT VAL("")            : REM = 0
100 PRINT VAL("1")+VAL("2"): REM = 3
```

Mint a fenti példák is mutatják, *bármi* is az argumentum értéke, a VAL függvény mindig előállít valamilyen számot. Ez gyakran lehet szemantikus hiba forrása. Utolsó példánk mutatja, hogy VAL aritmetikai függvény, és ezért tetszőleges aritmetikai kifejezésben használható.

A VAL felismeri a +, -, E jeleket, a tizedespontot (.), és ezenkívül természetesen a 0-9 számjegyeket. Az első - szám részeként - nem értelmezhető karakter befejezi a sztring kiértékelését.

Hibalehetőség: Nincs.

VERIFY

Rövidítés: vE *Token*:$95(149)

Mód: mind parancs, mind program módban használható.

Ellenőrzi, hogy valamely periférián elmentett program megegyezik-e a memóriában tárolt programmal.

Szintaxis: megegyezik a LOAD szintaxisával.

Példák:

```
(i)   SAVE "PROG",8
      VERIFY "PROG",8
(ii)  SAVE "PROG" : REM CSEVELD VISSZA
      VERIFY "PROG"
```

Az (i) példában a PROG nevű programot kiírjuk a lemezre, majd ellenőrizzük, nem történt-e hiba. A VERIFY parancs kiadása után a következő üzeneteket kapjuk:

SEARCHING FOR "PROG"
VERIFYING

Ezeket egy OK vagy VERIFY ERROR üzenet követheti, attól függően, hogy a lemezes file tartalma megegyezik-e a memóriában tárolt programmal vagy sem.

A (ii) példa ugyanezt a programot kazettás file-ba menti el, s ezt ellenőrzi. A VERIFY parancs kiadása előtt a szalagot a program elé kell visszatekerni. A VERIFY "PROG" parancs kiadása után a képernyőn a

PRESS PLAY ON TAPE

üzenet jelenik meg. A kazettás egység <PLAY> billentyűjének a lenyomása után megkezdődik a program megkeresése, majd az ellenőrzése.

Hibalehetőségek Ha programból hajtjuk végre a VERIFY parancsot, és az OK-val végződik, a program fut tovább. Ha hibát észlel az interpreter, ?VERIFY ERROR hibajelzést kapunk, és a program futása megszakad. Ha a program nincs a lemezen vagy a kazettán, az interpreter egy EOT fejlécet olvas, akkor ?FILE NOT FOUND ERROR hibajelzést kapunk.

VOL

Rövidítés: vO *Token*$DB(219)

Mód: mind parancs, mind program módban használható.

A hanggenerátor hangerejét a VOL-ban megadott értékre (0-7) állítja be. Ha ez 0, akkor a hanggenerátor egyáltalán nem szól.

Szintaxis: VOL <aritmetikai kifejezés>

Példák:

```
(i)     VOL 0
(ii)    10 VOL 7
        20 SOUND 2,600,50
(iii)   100 FOR I=0 TO 3
        110 VOL 7-I
        ....
        300 NEXT I
```

Az (i) példa kikapcsolja a C-16 hanggenerátorát. A (ii) zenélő programokban tipikus: maximumra kapcsoljuk a hanggenerátort. (iii) egy cikluson belül folyamatosan csökkenti a hangerőt.

Hibalehetőségek Nincs.

WAIT

Rövidítés: wA *Token*: $92(146)

Mód: mind parancs, mind program módban használható.

Az utasítás felfüggeszti a program végrehajtását, míg a paramétereiben specifikált esemény be nem következik.

Szintaxis:
WAIT <arit. kif.>, <arit. kif.> [<arit. kif.>]

Az első aritmetikai kifejezés a memória valamely címét jelenti, és így értékének a 0-65535 intervallumba kell esnie. A másik két aritmetikai kifejezés értékének a 0-255 intervallumba kell esnie.

A WAIT L,M1,M2 utasítás először kiszámítja a

(PEEK(L) EOR M2) AND M1

értékét. (EOR a **kizáró vagy.**) Ha az eredmény nullától különböző, a BASIC program folytatódik; ha 0, akkor az interpreter a fenti műveletet - akár a végtelenségig - ismétli. L-nek tehát olyan címnek kell lennie, amit a hardver megszakító rutin, vagy a perifériák használnak, és a benne tárolt értéket megváltoztathatják a kívánt módon. (Ha M2-t nem adtuk meg, akkor csak PEEK(L) AND M1 hajtódik végre.)

Az az esemény, amelyik a BASIC program futásának folytatását eredményezi az L cím valamely bitjének (bitjeinek) magasra és/vagy alacsonyra állítódása lehet. Az M1-ben magasra kell állítani azokat a biteket, amelyek értékére kíváncsiak vagyunk, a többit pedig alacsonyra. M2-ben azokat a biteket kell magasra állítani, amelyek értékének alacsonyra kell válnia a futás folytatásához. Ha például M1=00011000=24 és M2=00010000=16, akkor a program addig áll, míg az L cím 4. bitje alacsony és az 5. bitje magas nem lesz. Mihelyt a 4. bit magas vagy az 5.bit alacsony lesz, a program fut tovább.

Példák:
(i) 10 WAIT 239,255: GET A$: PRINT A$: GOTO 10

Az (i) példában a program addig vár, míg meg nem nyomunk egy billentyűt. Igy a GET A$ utasítás sohasem olvas CHR$(0) értéket. Hasonlítsuk össze a futás eredményét a 10 GET A$:PRINT A$:GOTO10 programmal! (Lásd a 4. fejezetben a billentyűzet-pufferről szóló részt!)

Hibalehetőség Nincs (de végtelen ciklusba kerülhet a program).

WHILE

Rövidítés: wH *Token*:$FD(253)

Mód: mind parancs, mind program módban használható.

A DO vagy LOOP utasításban levő kilépési feltétel megadására használhatjuk. A ciklus akkor jár le, ha a WHILE utáni logikai kifejezés hamissá válik.

Szintaxis:

```
 {DO  }
[{    }] WHILE <logikai kifejezés>
 {LOOP}
```

Példák:

```
(i)     DO WHILE JOY(1)>0: LOOP
(ii)    2300 DO
        2310 INPUT "<CLEAR>X=";X
        2320 LOOP WHILE X<0
```

(i)-ben a ciklus addig jár, amíg az 1-es botkormány nem kerül alaphelyzetbe. A (ii) alatti beolvasó rutin ciklusa addig jár, amíg a 2320-as sorban a WHILE utáni feltétel igaz, azaz X<0. A ciklus tehát az első nem negatív érték beírásakor jár le.

Hibalehetőségek A logikai kifejezés kiértékelése közben számos hiba történhet. További hibát okozhat az WHILE-t tartalmazó DO vagy LOOP nem megfelelő használata.

4. fejezet
A perifériák használata

4.1 Bevezetés

A mikroszámítógépek felhasználási lehetőségeit, teljesitőképességét elsősorban nem az határozza meg, hogy milyen műveleteket és hogyan végeznek, hanem az, hogy hogyan képesek 'környezetükkel' **kommunikálni.** A C-16 számítógépet úgy tervezték, hogy igen sokfajta eszközt tudjon kezelni. Ezek közül egyről, a **lemezegység**ről külön szólunk az 5. fejezetben. Most a további lehetőségeket ismertetjük az alábbi sorrendben:

a/ **kazettás egység**;
b/ **nyomtatók**;
c/ **botkormányok**;
d/ **billentyűzet.**

A korábbi Commodore gépek még a központi egységgel és a billentyűzettel egybeépitett kazettás egységet tartalmaztak (mint például a HT 1080Z számítógép). A lemezegység használatának elterjedésével azonban a kazettás egység kikerült a Commodore gépeiből. A C-16-hoz speciális kazettás egység, az úgynevezett **Datasette** csatlakoztatható. (A C-64-hez használható változat nem megfelelő, mert eltérő a csatlakozója.) A kazettás egység annyiban különbözik a háztartási magnetofonoktól, hogy a C-16 lekérdezheti, hogy van-e benyomott billentyű, vagy sem.

Az egység tetszőleges, legfeljebb 30 perc játékidejű, jó minőségű kazettával használható. A kazettás egység előnye olcsósága és a valódi tároló rész, a kazetta könnyű beszerezhetősége. A lemezegységgel szemben azonban csak **soros adattárolás** valósítható meg, és az is lényegesen lassabban.

4.2 Kazettás egység

A kazettás egységen hat billentyű és egy jelzőlámpa található. Ezek a háztartási magnetofonokon szokásos billentyűk. A kazettás egység és a számítógép közti adatátvitel aszinkron, ezért a szalag indításáról a kezelőnek kell gondoskodnia. A szalag megállítása már automatikus. A kazettás egység használata előtt a szalagot a megfelelő helyre kell csévélni. Erre szolgálnak a **<REWIND>** (hátra), illetve **<F.FWARD>** (gyorsan előre) billentyűk. További teendőinket a számítógép kiírja. Ha olvasunk a kazettá-

ról, akkor

PRESS PLAY ON TAPE

felirat jelenik meg a képernyőn. Válaszként meg kell nyomnunk a <**PLAY**> billentyűt, mire az olvasás elkezdődik.

Irás (felvétel) esetén a

PRESS PLAY & RECORD ON TAPE

felirat jelenik meg a képernyőn. Válaszként a <**PLAY**> és a <**RECORD**> billentyűket ***egyszerre*** kell megnyomnunk. A kazettás egységen levő lámpa kigyullad, jelezve, hogy írunk a szalagra. Az adatátvitel befejezése után a számítógép automatikusan megállítja a szalagot. Ezután nyomjuk meg a <**STOP**> billentyűt (a *kazettás* egységen!). Az <**EJECT**> gomb megnyomása után a kazetta kivehető. Ha adatokat tárolunk a kazettán, akkor a C-16 többször is megállítja, illetve elindítja a szalagot. Ügyeljünk arra, hogy csak az átvitel *teljes* befejezése után vegyük ki a szalagot.

Adattárolás kazettán
A C-16 a szalagon az adatokat négyszögimpulzusok formájában tárolja. Minden byte-ot egy bytejelző előz meg. Ezután következik a 8 bit, majd a végén a paritásbit (páros).

A C-16 két file típust kezel a kazettás egységen, ezek a lemezegység **PRG** és **SEQ** file-jainak felelnek meg. Mindegyik esetben először a file **fejléce** (header) kerül kiírásra, amelyik a file-ra vonatkozó információkat (név, kezdőcím stb.) tartalmazza. Ezt követik az adat byte-ok, programfile esetén egyetlen összefüggő blokkban, adatfile esetén pedig pufferenként egy blokkban.

Programfile-ok használata

A BASIC nyelvű programokat a LOAD és a SAVE utasítás segítségével olvashatjuk be, illetve írhatjuk ki a szalagra. A kazettás egység gyakran hibázhat, ezért a fontosabb programokat célszerű kétszer is felvenni (elmenteni). A programfile-ok eltérő tárolási módjuk miatt nem olvashatók szekvenciális file-ként (szemben a lemezen levő programfile-okkal)

Szekvenciális file-ok használata

Adatfile-okat a szalagon az OPEN utasítás segítségével nyitunk meg, illetve a CLOSE utasítás segítségével zárunk le. (Lásd a 3. fejezetet) Az OPEN utasítás hatására a fejléc a pufferba kerül,

és ott a PEEK utasítás segítségével megvizsgálható.

A szalagra a PRINT# utasítással írhatunk. A C-16 az adatokat először a kazetta-pufferba tölti, és csak a puffer megtelése után írja ki a szalagra. A CLOSE utasítás hatására még az esetleg a pufferban levő adatok kiíródnak, és csak utána kerül sor a file lezárására. A speciális karakterek egy része (pl. CHR$(13)) ugyancsak kiíródik a szalagra.

A szalagról adatokat mind a GET#, mind az INPUT# utasítás segítségével olvashatunk.

A fenti utasítások használatához azonban feltétlenül tudni kell, hogy a C-16 kazettás egységre bizonyos **speciális** karakterek **nem íródnak ki.** CHR$(10) (a soremelés karakter) nem kerül kiírásra, hasonlóan a CHR$(29) <CRSR JOBBRA> jelhez. A szalagon egy CHR$(0) jel a file végét jelenti, és ST=64 lesz. (Csak adatfile-ok esetén!) Az INPUT# használata közben a következő karakterek okozhatnak problémát: CHR$(13), CHR$(32), CHR$(34), CHR$(44) és CHR$(55).(Ezek sorrendben <RETURN>, szóköz, idézőjel, kettőspont, vessző).

A GET# utasítás hatása egyszerű: az adatfile következő karakterét olvassa. Az INPUT# az input sort a kazettás egységről a legközelebbi CHR$(13) karakterig olvassa be. Az input sor legfeljebb 88 karakter hosszú lehet. Amennyiben az INPUT# utasításban több változó is szerepel, a megfelelő elválasztó jelekről (általában vesszőről) a szalagra való íráskor kell gondoskodni. Ha például három számot az INPUT#1,A,B,C utasítással szeretnénk visszaolvasni, akkor ezeket a PRINT#1,A;X$;B;X$;C utasítás segítségével kell kiírnunk, ahol X$=",". (A PRINT# automatikusan gondoskodik a CHR$(13) kiírásáról, ha a változólista nem vesszővel vagy pontosvesszővel végződik.)

Egy egyszerű példát adunk, amelyik először a billentyűzetről beolvas 20 adatot, majd ezeket kiírja a szalagra. A program második része az adatokat visszaolvassa a szalagról és kiírja a képernyőre. A beolvasáskor egy INPUT utasítással 2 adatot szeretnénk beolvasni. Ezért a szalagra való íráskor a két érték közé nekünk kell a vesszőt kiírnunk:

```
100 REM ADATOK BEOLVASASA
110 DIM A(12):X$=","
120 FOR I=1 TO 12
130 PRINT I;". HONAP"
140 INPUT "TERMELES=";A(I)
150 NEXT I
160 :
170 REM KIIRAS A SZALAGRA
180 OPEN 7,1,1,"ADATOK"
```

```
190 FOR I=1 TO 12
200 PRINT#7,I;X$;A(I)
210 NEXT I
220 CLOSE 7
230 :
240 REM VISSZAOLVASAS
250 PRINT "VISSZACSEVELTED A SZALAGOT?"
260 GET A$: IF A$<>"I" THEN GOTO 260
270 :
280 OPEN 7,1,0,"ADATOK"
290 FOR I=1 TO 12
300 INPUT#7,K,B
310 PRINT K;". HONAP TERMELESE";B
320 NEXT I
330 CLOSE 7
```

4.3 Nyomtatók

A legtöbb mikroszámítógépes rendszer tartozéka az adatok vizuális megjelenítésére szolgáló nyomtató. A nyomtató a képernyőhöz nagyon hasonló output eszköz. Használata közben arra kell ügyelnünk, hogy a nyomtatóra küldött vezérlő karakterek könnyen értelmezhetők legyenek. Például a <szóköz> jel a nyomtatón azonos módon használható, mint a képernyőn. Ugyanakkor a TAB függvény nem ugyanúgy működik, mivel a sorban az aktuális pozíció a képernyőre és nem a nyomtatópapírra vonatkozik. Hasonlóan a <CRSR JOBBRA> vagy a <CLEAR> karaktereknek egészen más a jelentése.

Hazánkban a C-16-hoz szinte kizárólag mátrixnyomtatókat használnak, amelyek a C-16 **mindkét karakterkészletét** nyomtatják. További speciális funkciókat is ellátnak, amelyek többé-kevésbé gépfüggetlenek. Használatuk előtt ezért a nyomtatóhoz szállitott gépkönyvet mindenképp célszerű áttanulmányozni.

A nyomtató a soros buszra kapcsolódik. A pufferje megtöltéséig folyamatosan fogadja az adatokat, majd a nyomtatás idejére letiltja az adatküldést. Ha ilyenkor adunk ki egy, a nyomtatóra vonatkozó PRINT# utasitást, a BASIC addig vár, míg nem lehet további adatokat küldeni.

A nyomtatók hardver egységszáma 4 vagy 5 lehet. A gyártó cégek általában 4-re állitják be a nyomtatót. Egyes nyomtatók egységszámát a hátán vagy oldalán levő kapcsolóval lehet 5-re átállítani.

A nyomtató megnyitása
Szintaxis: OPEN lf,<egységszám>,<mód>

Az első kiíró utasítás kiadása előtt meg kell nyitni a nyomtatót. Az utasítás fenti alakjában lf a ***logikai file*** szám, amit a PRINT# és CMD utasításokban lehet használni. Az <egységszám> 4 vagy 5 lehet. A <mód> sztring parancsként a soros buszra kerül. Jelentése a kiválasztott nyomtató típusától függ. A C-16-hoz gyárilag szállított nyomtatók esetén a két legfontosabb érték:

0	nagybetűk/grafikus jelek
7	kisbetűk/nagybetűk

Irás a nyomtatóra

Irásra a PRINT# és a CMD utasításokat használhatjuk. A nyomtatási kép az elküldött vezérlő karaktereknek megfelelően alakul. A legfontosabb vezérlő karakterek a következők:

CHR$	**Jelentés**
10	Soremelés
13	<RETURN> (automatikus soremelés a CMB nyomtatókon)
14	Dupla vonalvastagság-mód beállítása
15	Dupla vonalvastagság-mód törlése
18	Inverz karakter-mód beállítása
146	Inverz karakter-mód törlése
17	Kisbetűk/nagybetűk karakterkészlet
145	Nagybetűk/grafikus jelek karakterkészlet
16	Tabulálás

Példák (OPEN 4,4 után):

```
PRINT#4,CHR$(17);"ABRAKADABRA": REM KIS BETUK
PRINT#4,CHR$(27);CHR$(10);"*": REM A 10. POZICIóRA IRJA A *-T
```

A nyomtató lezárása

A nyomtatóra - lezárás előtt - még egy üres PRINT utasítást kell küldeni, hogy a nyomtató pufferében esetleg még bent levő karakterek is kiíródjanak. Igy a

```
PRINT#lf: CLOSE lf
```

utasítás zárja le helyesen az lf logikai file számmal megnyitott nyomtatót.

4.4 Botkormányok

A C-16 két nyolc érintkezős játék I/O csatlakozóval rendelkezik, amelyek lehetővé teszik két botkormány csatlakoztatását. A digitális botkormány öt különálló kapcsolóval rendelkezik. Négy kapcsoló szolgál az irányításra, egy pedig tüzelő billentyűként használható. A botkormány kapcsolói az alábbi módon vannak elrendezve:

```
      TŰZ                                 FEL
(4-es kapcsoló)                     (0-ás kapcsoló)



           BALRA                                    JOBBRA
       (2-es kapcsoló)                          (3-as kapcsoló)



                                          LE
                                    (1-es kapcsoló)
```

A botkormány kapcsolóinak állásáról a JOY BASIC utasítás segítségével kaphatunk információt. Ez nagy könnyebbség a C-64-hez, vagy a VC-20-hoz képest, ahol speciális programokat kellett írni a botkormányok használatához. Az alábbi programban a botkormány segítségével mozgathatunk a képernyőn keresztül egy autót.

```
100 REM ADATOK BEOLVASASA
110 FOR I=1 TO 60: READ X: A$=A$+CHR$(X): NEXT I
120 FOR I=1 TO 8: READ DX(I):NEXT I
130 FOR I=1 TO 8: READ DY(I):NEXT I
140 :
150 COLOR 0,14,5
160 COLOR 4,14,5
170 GRAPHIC1,1
180 :
190 X=290:Y=170: GSHAPE A$,X,Y,2
200 :
```

```
210 DO
220 M=JOY(1): IR=M AND 127: S=(M AND 128)/128+1: IF IR=0 THEN
GOTO 280
230 XU=X+DX(IR)*S: XU=XU-INT(XU/319)*319
240 YU=Y+DY(IR)*S: YU=YU-INT(YU/199)*199
250 GSHAPE A$,X,Y,4
260 GSHAPE A$,XU,YU,4
270 X=XU:Y=YU
280 GETKEY U$
290 LOOP UNTIL U$=CHR$(13)
300 :
310 GRAPHIC 0: END
320 :
330 REM AZ AUTO ADATAI
340 DATA
0,0,0,0,0,8,0,0,0,31,248,0,0,99,12,0,1,131,12,0,7,3,6,0,12,3,6,0
350 DATA
255,255,255,0,255,243,255,0,255,255,255,0,109,255,182,0,30
360 DATA 0,120,0,30,0,120,0,12,0,48,0,24,0,13,0
370 REM AZ IRANYOK ADATAI
380 :
390 DATA 0,3.5,5,3.5,0,-3.5,-5,-3.5
400 DATA -5,-3.5,0,3.5,5,3.5,0,-3.5
```

4.5 **Billentyűzet**

A C-16 billentyűzete a központi egységgel összeépítve kerül forgalomba. Az interpreter egy **tíz karakter hosszú billentyűzet-pufferrel rendelkezik**, amiben a leütött, de még fel nem használt karakterek vannak. A billentyűzet olvasását a hardver megszakító rutin végzi. Ez ellenőrzi a <STOP> billentyű megnyomását. Ugyancsak ez a rutin végzi a funkció billentyűk megnyomásának ellenőrzését. Ha egy funkcióbillentyűt megnyomunk, akkor a karaktereket nem a billentyűzet-pufferből, hanem egy kölün területről veszi az interpreter, ahol a KEY parancs segítségével tárolt szövegrészek vannak.

A billentyűzethez tartozó legfontosabb memóriahelyek a következők:

Cím	**Jelentés**
239	a billentyűzet-pufferben levő karakterek száma
1319-1328	a billentyűzet puffer
1343	a billentyűzet-puffer hossza
1344	ismétlés módja

A billentyűzet-pufferbe a POKE utasítás segítségével mi magunk is vihetünk be karaktereket, úgy mintha a billentyűzetről írnánk be. Ezzel elérhetjük például azt, hogy a program egy END végrehajtása után 'magától' újraindul. Az alábbi programrész kilistázza a programot a nyomtatóra, majd újraindítja azt, de most már listázás nélkül:

```
10 PRINT CHR$(147);"PRINT#1:CLOSE1:GOTO50
20 POKE 1319,19: POKE 1320,13
30 POKE 239,2
40 OPEN 1,4: CMD 1: LIST
50 <további utasítások>
```

A 10-es sorban kiírjuk a képernyő tetejére a

PRINT#1: CLOSE 1: GOTO 50

parancsot. A 20. sorban a billentyűzet pufferbe írjuk a <HOME> és a <RETURN> karaktereket, a 30. sorban pedig jelezzük, hogy két karakter van a pufferban. A listázás befejezése után az interpreter végrehajtja a <HOME> és <RETURN> billentyűk megnyomásának megfelelő tevékenyységet, s végrehajtja a képernyő első sorában látható parancsot.

A szövegszerkesztő használatánál már említettük, hogy minden billentyű ismétli önmagát. A szövegszerkesztőnek ezt a tulajdonságát az 1344-es memóriahely tartalmától függően a következőképpen lehet módosítani:

0	csak a kurzor vezérlő karakterek ismétlődnek,
64	semelyik karakter sem ismétlődik,
128	valamennyi karakter ismétlődik.

5. fejezet

A VIC 1541-es lemezegység

5.1 A lemezegység felépítése

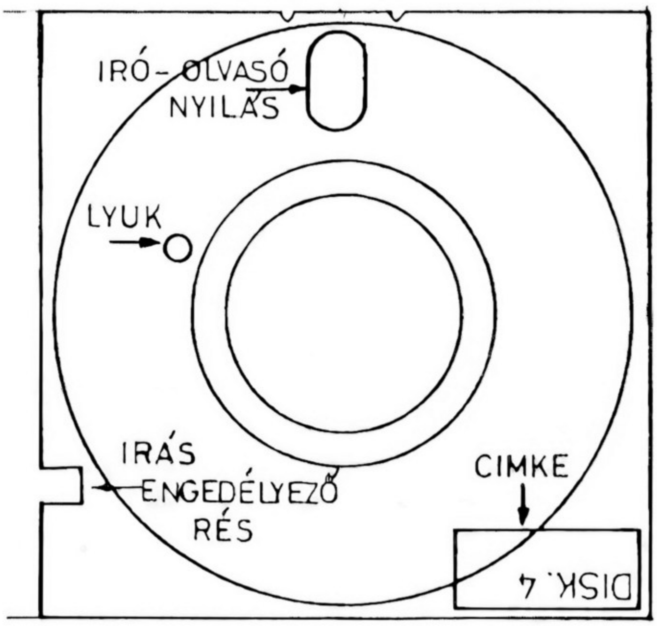
Ebben a fejezetben a C-16 számítógéphez csatlakoztatható VIC-1541 tipusú lemezegység használatát ismertetjük.

A legtöbb lemezegység - így a VIC-1541 is - egy író/olvasó, illetve egy törlő fejjel rendelkezik, amelyeket egy léptető motor segítségével lehet a lemez megfelelő sávja fölé pozicionálni. A motor mindig sugárirányban mozgatja a fejeket, egy-egy lépés nagysága megközelítőleg 1/40 inch. A lemezen annak a sávnak a szélessége, amelyen az írás végül is történik, körülbelül 1/80 inch. A lemezegység motorja nagy sebességgel forgatja a lemezt, amelyik a centrifugális erő hatására elveszti hajlékony jellegét, és így lehetővé válik az író/olvasó fej mozgatása.

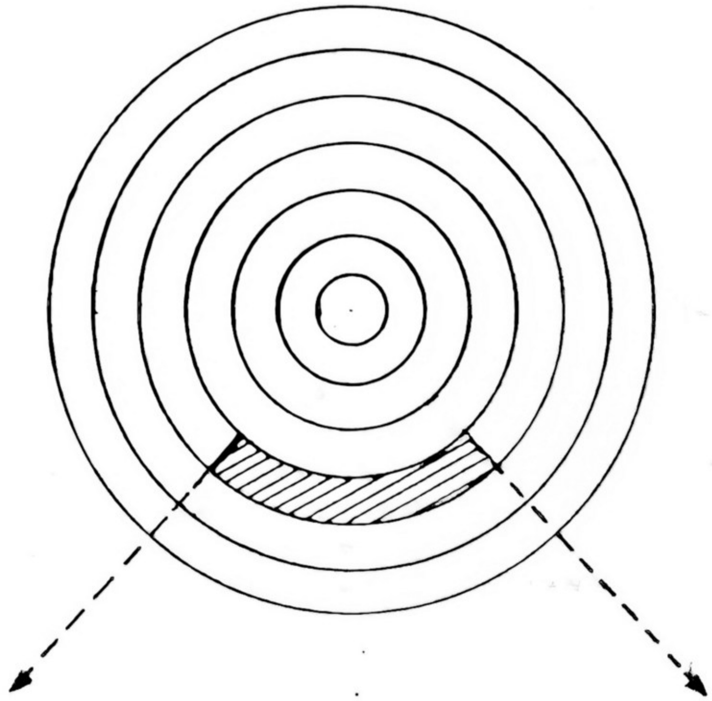
A lemezegység számos áramkört, érzékelőt tartalmaz, amelyek lehetővé teszik a léptető motor vezérlését, az írást engedélyező rés meglétének ellenőrzését, a lemezegység adott sávjának olvasását stb. A VIC-1541-es lemezegység egy önálló 16K-s **operációs rendszert(DOS)** tartalmaz, amelyik a lemezegység kezelésén túlmenően a C-16-tal való kommunikációt is elvégzi. A DOS használata nagyfokú önállóságot eredményez, ami lehetővé teszi, hogy a központi egységtől függetlenül is dolgozhasson a lemezegység. Elérhető például, hogy a lemezegység egy file-t a sornyomtatón listázzon, miközben a központi egység valami más feladatot végez. A számítógép csatlakozója úgy lett megtervezve, hogy egyszerre négy lemezegység és egy nyomtató legyen rácsatlakoztatható. (Ebben az esetben természetesen a második, illetve a további lemezegységek hardver számát - ami általában 8 - módosítani kell.)

A C-16 BASIC interpreter a korábbi Commodore gépekhez képest számos, a lemezegység kezelését megkönnyítő BASIC utasítást tartalmaz. Természetesen ezeket célszerű használni a bonyolultabb DOS parancsok helyett. Ha azonban olyan programot szeretnénk írni, amelyik valamennyi Commodore gépen fut, akkor nem használhatjuk ezeket az utasításokat.

Az alábbi ábra egy hajlékonylemez (floppy) vázlata. A négyzet a lemezt magában foglaló borítékot szemlélteti, a kör alakú rész a mágneses adathordozó valódi (de nem látható) szélét mutatja. Az írásvédő rés a lemezek fizikai védelmét szolgálja. Amennyiben az írásvédő rést **kivágtuk**, a DOS nem akadályozza meg a lemezre való **írást**. Ha a rés **hiányzik** (pl. leragasztottuk), a DOS **nem engedi** meg a lemezre való írást.



A DOS a lemezre **sávonként** (track) írja az információt. A sávok koncentrikus körgyűrűk. A lemezen 35 sáv helyezkedik el, amelyeket kívülről 1-gyel kezdődően számoz meg a rendszer. A sávra szeletenként, szektoronként (sector) egy blokknyi adat (256 byte) információ kerül felírásra úgy, ahogy ezt az alábbi ábra szemlélteti:



Egy szektorba a következő információk kerülnek:

SYNC	08	ID1	ID2	TRACK	SECTOR	ELL.OSSZEG	GAP1

SYNC	07	256 byte adat	ELL.OSSZEG	GAP2

A DOS rendszer automatikusan gondoskodik a lemezre való írásról/olvasásról. A Commodore cég lemezegységei - eltérően a legtöbb gyártótól - a külső (alacsonyabb sorszámú) sávokban sűrűbben írnak, és így ott több szektor fér el:

sáv(track) sorszáma	**szektorok indexe**	**szektorok száma**
1-17	0-20	21
18-24	0-18	19
25-30	0-17	18
31-35	0-16	17

A fenti adatokból is látható, hogy a lemezre összesen 683 blokknyi információ fér el. Ezek közül a DOS 19 blokkot szervezési célokra használ, ezért egy lemez tárolókapacitása 168566 byte (664 blokk).

A lemezegység és a lemezek megfelelő karbantartása esetén egy bit meghibásodásának a valószínűsége igen kicsiny (kb. 1E-10). A DOS rendszerhibák ennél gyakoribbak. A DOS, ha hibát észlel, az író/olvasó fejet a lemez széléig lépteti, majd újból megkísérli az olvasást. Ezt az eljárást tízszer ismétli, és csak azután ad hibajelzést.

5.2 **Tárolási alapelvek**

A DOS a lemezen tárolt adatokról nyilvántartást (katalógus) vezet, ami rögziti, hogy a lemez mely sávjai és szektorai tartalmaznak információt; milyen programokat, adatfile-okat tárolunk a lemezen stb. A DOS két módot kínál a lemezre való írásra/olvasásra. Az első esetben a lemez **katalógus**ában (**directory**) szereplő file-okat használjuk, a második esetben közvetlenül az általunk kiválasztott blokkba írunk (vagy onnan olvasunk ki) információt.

A DOS a lemezen tárolt file-ok kezeléséhez szükséges információt a 18. sáv 0. szektorában kezdődő katalógusban tárolja. Ezek az információk a következők:

a/ a lemez neve és azonosító száma;
b/ a lemez formája;
c/ minden egyes, a lemezen levő file
- neve
- típusa
- blokkjainak a száma
- törölt-e vagy sem?
- védett-e?
- a file első blokkjának sáv/szektor száma

A katalógusra a különböző utasításokban a $ jellel lehet hivatkozni. A katalógus egy speciális program file-nak is tekinthető, amelyik például a LOAD"$",8 utasítással betölthető a memóriába, majd a LIST paranccsal kilistázható. A blokkok foglaltságának jelzésére 140 byte szolgál a katalógus elején. A katalógusnak ezt a részét **BAM**-nak hívjuk az angol elnevezés (**block availability map**) alapján.

A C-16 számítógép a DIRECTORY BASIC utasítással lehetőséget nyújt arra, hogy a katalógust a képernyőre listázzuk. Erre főleg akkor van szükség, ha a memóriában tárolt programot nem akarjuk megsérteni.

File-ok azonosítására a nevük szolgál. Normál körülmények közt egy lemezen két azonos nevű file akkor sem lehet, ha különböző a típusa. A file-ok nevét a BASIC utasításokban rövidíthetjük. Ha egy file név *-ra végződik, akkor azt a lemezegység az azzal a névvel kezdődő első file-lal azonosítja. A névben szereplő kérdőjelek helyén tetszőleges karakter állhat.

Az ugyanahhoz a file-hoz tartozó blokkokat a DOS összeláncolja. Ez azt jelenti, hogy a blokk első két byte-ja mindig a file következő blokkjának sáv és szektor számát adja meg. A file-hoz tartozó első blokk adatait a katalógusban találjuk meg. Mivel 0. sorszámú sáv nincs, ezért a 0. sorszámú sáv a file utolsó blokkját jelenti. Ebben az esetben a 2. byte az adott blokkon belül az utolsó, **még** a file-hoz tartozó byte sorszámát jelenti.

Az ugyanahhoz a sávhoz tartozó blokkok foglaltságát 4 byte jelzi. Az első byte a sáv (track) szabad byte-jainak számát adja meg, az azt követő 24 bit pedig az egyes blokkok foglaltságát jelzi. A 0 bit foglaltat, az 1 bit szabadot jelent. A nem létező szektoroknak megfelelő utolsó bitek mindig nullák.

5.3 A 15. csatorna használata

A lemezzel történő adatcseréhez a file megadásán túl két további adatot kell megadnunk, amelyek mindegyike technikai jellegű, és az adatcserét nem befolyásolja. A BASIC kiíró/beolvasó utasításai az egyes file-okra egy-egy számmal hivatkoznak, amelyeket a file megnyitásakor kell megadnunk. Ezek a **logikai file szám**ok, amelyek értéke az 1-255 intervallumba kell hogy essen. A 0-ás file szám használatát a BASIC szintaktikus hibának tekinti. A másik ilyen - a ki/beviteli műveletet - szabályozó technikai adat a **csatornaszám**, aminek az értéke a 0-15 intervallumba kell hogy essen. A csatornaszámot a lemezegység használja a file-ok azonosítására. A file nevére csak a megnyitásakor van szükség, attól kezdve a logikai file számot, illetve a csatornaszámot használhatjuk az azonosítására (meghatározott és a későbbiekben ismertetett szabályok szerint).

A DOS a 0., 1. és 15. csatornákat speciális célokra használja. A 0. és 1. csatornákat a SAVE és LOAD utasítások használják. Adatok továbbítására a 2.-14. csatornákat használhatjuk. A 15. csatornát **hiba** vagy **parancs csatorná**nak hívjuk. Ezen keresztül adhatunk parancsokat a DOS-nak, illetve kaphatunk információt a lemez állapotáról. Ha a lemezegység hibát észlel, azt az előlapon látható **piros lámpa villogása** jelzi. Előfordulhat, hogy a BASIC ***nem*** ad hibajelzést. Ilyenkor csak a hibacsatorna olvasásával kaphatunk információt arról, milyen hiba is történt.

A 15. csatornát általában az **OPEN 15,8,15** utasítással nyitjuk meg. Ennek hatására a BASIC nyilvántartásba vesz egy 15-ös file számu file-t, amelyik a lemezegység 15. csatornáját használja. Ha a PRINT# utasítással ebbe a file-ba írunk, az nem íródik a lemezre, hanem a DOS parancsnak tekinti és végrehajtja.

Lemezek formázása
Valahányszor egy új, még eddig nem használt lemezt akarunk használni, a lemezt először formázni kell. A DOS ilyenkor elhelyezi a megfelelő szinkronizációs jeleket, felírja az üres katalógust stb. Az utasítás alakja:

```
PRINT#15,"NEW:<név>,id"    vagy rövidítve:
PRINT#15,"N:<név>,id"
```

A <név> karaktersorozat a lemez neve lesz. A név csak a katalógusba kerül beírásra, mig az id **két karakteres azonosító** valamennyi blokkba. Ha két kiíró utasítás között kicseréljük a

lemezt, a DOS érzékeli, hogy új id azonosítójú lemezzel dolgozik és hibát jelez. A fenti utasítás PRINT#15,"N:<név>" alakja **csak** a katalógust törli, de nem formázza újra a lemezt, és csak a nevet változtatja meg.

A NEW DOS-parancs természetesen használt lemezekre is kiadható, de ebben az esetben a rajta levő információ teljes egészében elvész. Használata előtt tehát mindig győződjünk meg róla, hogy a rajta levő adatokra már tényleg nincs szükségünk.

Ugyanezt a feladatot elvégezhetjük a HEADER BASIC utasítás segítségével is. Ebben az esetben nem kell a 15. csatornát megnyitni, a BASIC interpreter maga gondoskodik mindenről.

A lemezegység inicializálása
Elsősorban DOS rendszerhiba vagy BASIC programhiba után előfordulhat, hogy a lemezegység pufferében tárolt BAM nem egyezik meg a lemezen levő BAM-mal. Ez meglevő file-ok felülírását eredményezheti. Ilyen esetben célszerű a lemezegységet inicializálni, melynek hatására a DOS lezárja a megnyitott file-okat és újra olvassa a BAM-ot.

```
PRINT#15,"INITIALIZE" vagy rövidítve:
PRINT#15,"I"
```

File-ok törlése
Lehetőség van egy vagy több file törlésére. A törlés nem jelenti a fizikai törlést, csupán a katalógusban jelzi egy bit, hogy a szóban forgó file már nem létezik. Amikor legközelebb egy új file-t hozunk létre, az írja felül a lemez megfelelő részeit. Az utasítás alakja:

```
PRINT#15, "SCRATCH:<név>"  vagy
PRINT#15, "S:<név>"
```

A névben szereplő * illetve ? karakterek jelentése speciális. A * karakter csak a név utolsó karaktere lehet. Ebben az esetben az összes, azzal a névvel kezdődő file törlődik. Ha a

```
PRINT#15,"S:GRID*"
```

utasítást adjuk ki, akkor törlődnek a következő programok: GRID, GRIDBOOT, GRIDRUNNER stb. A kérdőjel azt jelenti, lényegtelen, hogy azon a helyen milyen betű áll. Ha a katalógus tartalmazza a PEK, POK, PIK nevű programokat, akkor a PRINT#15, "S:P?K" valamennyit törli.

A törlésre külön BASIC parancs szolgál, amit ugyancsak SCRATCH-nek hívnak. Használatához ugyancsak nem kell megnyitni a 15. csatornát, a BASIC interpreter ezt maga elvégzi.

A lemez újraszervezése

Ha egy lemezt már sokat használtunk, akkor az ugyanahhoz a file-hoz tartozó blokkok össze-vissza helyezkednek el a lemezen, és a DOS igen lassan tudja csak olvasni őket. Lehetnek a lemezen az OPEN utasítással megnyitott, de szabályosan le nem zárt file-ok is. A lemez rendbetételére a

```
PRINT#15,"VALIDATE"  vagy röviditve a
PRINT#15,"V"
```

DOS parancsot használhatjuk. Az utasítás hatására a DOS újraszervezi a lemezeken levő adatokat, anélkül, hogy azok tartalmát megváltoztatná. A VALIDATE DOS-parancs törli a véletlen-file blokkjait, ezért olyan lemez esetén, amelyik tartalmaz véletlen-file-t is, semmiképp se használjuk.

Ugyanez a feladat oldható meg a COLLECT BASIC utasítás segítségével.

File-ok átnevezése

Néha szükség lehet arra, hogy egy file-nak megváltoztassuk a nevét. Ezt a következő utasítással érhetjük el:

```
PRINT#15,"RENAME:<új név>=<régi név>"
PRINT#15,"R:<új név>=<régi név>"
```

File-ok másolása és összefűzése

Lehetőségünk van egy file másolatát ugyanazon a lemezen, de más név alatt létrehozni:

```
PRINT#15,"COPY:<új név>=<régi név>" vagy
PRINT#15,"C:<új név>=<régi név>"
```

A COPY DOS-parancsot maximum négy file összefűzésére is használhatjuk. A COPY ebben az esetben csak szekvenciális file-okat tud összefűzni:

```
PRINT#15,"C:<új név>=<r.név1>,<r.név2>,<r.név3>,<r.név4>"
```

Ugyanezt a feladatot oldja meg a COPY BASIC utasítás. A file-ok összefűzésére azonban csak a DOS-t használhatjuk.

A hibacsatorna olvasása

A 15. csatorna az INPUT#15,A$,B$,C$,D$ utasítással olvasható. Az A$, B$, C$, D$ értékei a DOS hibaüzenetét tartalmazzák. Az egyes változók jelentése:

A$=a hiba számkódja;
B$=a hiba megnevezése;
C$=a sáv,
D$=a szektor száma, amihez a hiba kapcsolódik.

Az A$, C$, D$ csak számjegyeket tartalmaz, ezért az INPUT utasításban aritmetikai változókkal helyettesíthetjük őket:

```
INPUT#15, E1,E2$,E3,E4
```

Ugyanezt végzi el a BASIC interpreter, amikor a DS vagy DS$ fenntartott változókra hivatkozunk. Kiolvassa a hibacsatornát, majd annak megfelelően beállítja DS, illetve DS$ értékét. A C-16 esetében a fenti utasításokat általában nem használjuk.

5.4 Programok tárolása

Programok lemezen való tárolására, illetve visszatöltésére speciális utasítások állnak rendelkezésre. A DOS a tárolásra és betöltésre a lemezegység 0., illetve 1. csatornáját használja, ezért ezeket adatcsatornaként nem lehet üzemeltetni.

A programfile-ok első két byte-ja a **töltési cím**, a program első karakterének a helye a memóriában. A file többi byte-ja (karaktere) a program része.

Programok betöltése

A lemezen tárolt programoknak a memóriába való betöltésére a LOAD utasítás szolgál. Alakja:

```
LOAD <név>,<egységszám> [,<mód>]
```

A <név> a **program nevét** definiáló sztringkifejezés; az <egységszám> a lemezes egység sorszáma, ami általában 8. A <mód> értéke 0 vagy 1 lehet, (ha nem írjuk ki, az 0-nak felel meg). <mód>=1 esetén a program a memóriába pontosan oda íródik vissza, ahonnan a SAVE utasítással elmentettük, vagyis a töltési címtől kezdődően töltődik be a memóriába. <mód>=0 esetén a program a BASIC munkaterület első poziciójától kezdődően töltődik be.

A katalógus a LOAD"$",8 utasítással tölthető be, majd LIST utasítással írható ki a képernyőre.

A LOAD részletes ismertetése a 3. fejezetben található meg. (Lásd a 3.3 fejezetben még a DLOAD BASIC utasítást is!)

Programok elmentése
A memóriában tárolt programokat a SAVE utasítás segítségével menthetjük ki lemezre. Az utasítás formája:

SAVE <név>, <egységszám>

A <név> és <egységszám> jelentése ugyanaz, mint a LOAD utasításban. Az utasítás a BASIC programot másolja ki a lemezre, ahonnan azután a LOAD utasítás segítségével visszatölthető. Nincs mód a memória valamely más részének az elmentésére. (Lásd még a DSAVE utasítást a 3.3 fejezetben.)

Az utasítás - feltéve, hogy <név> nevű file még nincs a lemezen -, létrehoz egy új, <név> nevű file-t, és a memóriában tárolt program tartalmát elmenti a file-ba. Ha a file már létezik, a mentés nem történik meg. Hibajelzést nem kapunk, csak a lemezegység piros lámpája villog. Ezt elkerülendő a SAVE első paraméterét

"@:<név>"

alakban kell megadnunk. Ennek hatására a DOS először törli a meglevő file-t, majd a SAVE normálisan végrehajtódik.

Programok ellenőrzése

A **VERIFY <név>, <egységszám>** utasítás ellenőrzi, hogy a <név> nevű file tartalma megegyezik-e a C-16 memóriájában tárolt programmal. Ha igen, akkor OK választ kapunk, ha nem, hibajelzést.

A programfile-ok a szekvenciális file-okhoz hasonlóan írhatók/olvashatók. Az egyes utasítások alakja és hatása megegyezik a szekvenciális file-oknál használhatókkal. Ha program file-t magunk írunk, akkor természetesen nekünk kell gondoskodni, hogy az olyan alakú legyen, amit a BASIC interpreter a LOAD utasítás végrehajtása után a memóriában elvár.

Példák
Szükségünk lehet egy lemezen tárolt program kezdőcímének ismeretére. Mint már említettük, a kezdőcím a megfelelő file első két byte-ja. Ennek megfelelően a következő program egy tetszőleges program file kezdőcímét állítja elő:

```
10 INPUT"FILE NEV";N$
20 OPEN 2,8,2,N$+",PRG,READ"
30 GET#1,X$: IF X$="" THEN X$=CHR$(0)
40 GET#1,Y$: IF Y$="" THEN Y$=CHR$(0)
50 X=ASC(X$): Y=ASC(Y$)
60 PRINT "KEZDOCIM="; X+256*Y
70 CLOSE 1
```

Program file-ok közvetlen írásával és olvasásával elérhetjük programrészek egymáshoz fűzését. A következő rövid program ezt a feladatot végzi el. A programunk a nagyobb gépeken meglevő APPEND utasítást szimulálja.

```
5 INPUT "ELSO FILE NEVE=";E$
6 INPUT "MASODIK FILE NEVE=";M$
7 INPUT "EREDO FILE NEVE=";F$
10 OPEN 2,8,2,E$+",P,R":OPEN 3,8,3,F$+",P,W"
20 GOSUB 900,ME=NE:GOSUB 1100:EL=0:MUT=NE
30 PRINT"KEZDOCIM=";NE:GOSUB 2000
40 CLOSE 2:OPEN 2,8,2,M$+",P,R"
50 GOSUB 900:EL=MUT-NE:MUT=NE:GOSUB 2000
60 PRINT#3,CHR$(0);CHR$(0);:CLOSE2:CLOSE 3:END
900 GOSUB 1000:Y$=X$:GOSUB 1000:NE=256*ASC(X$)+ASC(Y$):RETURN
1000 GET#2,X$:IF X$="" THEN X$=CHR$(0)
1010 RETURN
1100 Y$=CHR$(ME/256):X$=CHR$(ME AND 255):PRINT#3,Y$;X$;:RETURN
1200 FOR I=1 TO DB:GOSUB1000:PRINT#3,X$;:NEXT I:RETURN
2000 GOSUB 900:IF NE=0 THEN RETURN
2010 DB=NE-MUT-2:MUT=NE:ME=MUT+EL
2020 GOSUB 1100:GOSUB 1200:GOTO 2000
```

5.5 Adatok tárolása

Adatok tárolására két file-típust használhatunk. Ezek egyike a **szekvenciális** (SEQ) file-típus, amelyik a kazettás egységen levő SEQ file típussal egyezik meg. Ennél a típusnál lényegesen jobban használhatók a **relatív file**-ok (REL). Ez a file-típus lehetővé teszi a rekordok közvetlen írását/olvasását.

A DOS még egy ún. felhasználói file-típus használatát is biztosítja. A felhasználói file-okat a DOS 1.X verziójú változatai használták, és többé-kevésbé ugyanazt a feladatot látták el, mint a DOS 2.X verziójú változataiban a relatív file-ok. A DOS 2.6 a felhasználói file-okat gyakorlatilag nem különbözteti meg a szekvenciális file-októl.

5.5.1 Szekvenciális file-ok

A szekvenciális file-ok az adattárolás egyik legegyszerűbb formáját jelentik. A file-ba írt adatokat csak a felírás sorrendjében tudjuk visszaolvasni. Nincs mód a file adatainak módosítására. A szekvenciális file-t legegyszerűbben karakterek egymás utáni sorozatának képzelhetjük el:

```
K U T Y A F U L E , M A C S K A. . . . . . .
                    ↑

              író/olvasó fej
```

A sorozat valamelyik helyére pozicionálva áll az író/olvasó fej. Az írás és olvasás hatására a fej előre mozog. A file elejére csak úgy tudunk visszatérni, ha lezárjuk, és újra olvassuk.

A szekvenciális file-okkal a következő műveleteket végezhetjük:

a/ nyitás/zárás;
b/ írás vagy olvasás (kizáró értelemben).

Szekvenciális file megnyitása
Mielőtt a file-t használnánk, a megfelelő OPEN utasítás segítségével meg kell nyitnunk. Ennek alakja:

OPEN lf,<egység>,<csatorna>,"<név>,SEQ,<mod>"

lf a file logikai file száma. A továbbiakban ezzel az értékkel hivatkozhatunk a file-ra. Az <egység> a lemezegység **hardver száma**, általában 8. A <csatorna> a lemezegység 2-14 sorszámu **adatcsatornáinak** valamelyike. A <név> a **file neve**, SEQ pedig a típusa. <mód> a file megnyitási **mód**ja, lehetséges értéke: READ, WRITE illetve APPEND. A fenti szavak első betűjükkel rövidithetők. A <mód> értékétől függően a szekvenciális file-t

i/ **írásra** (WRITE);
ii/ **olvasásra** (READ);
iii/ **továbbírásra** (APPEND)

nyithatjuk meg. Az i/ esetben a lemezen az adott nevű file nem létezhet. Ha létezik, és egy ugyanolyan nevű új file-t szeretnénk létrehozni, akkor a "@:<név>,SEQ,WRITE" alakot kell használnunk. Ez törli a régi file-t és egy újat hoz létre ugyancsak <név> névvel.

Az ii/ esetben a lemezen értelemszerűen léteznie kell egy <név> nevű szekvenciális file-nak, amit olvasásra megnyitunk. Ha nem létezik, hibajelzést kapunk.

A iii/ esetben egy már létező szekvenciális file **írását folytatjuk.** Ennek megfelelően a file-nak léteznie kell a lemezen.

Az alábbi program bemutatja az egyes megnyitási módok hatását:

```
100 REM *********************
105 REM * DSK.SEQV2         *
110 REM *********************
115 REM * SZEKVENCIALIS FILE *
120 REM * MEGNYITASI MODOK   *
125 REM *********************
130 PRINT"s";
135 :
140 REM IRAS
145 REM ----
150 OPEN 2,8,2,"@:PROBA1,S,W" : PRINT "W";
155 FOR I=1 TO 3
160 PRINT#2,I : PRINT TAB(2);I
165 NEXT I
170 CLOSE2
175 :
180 REM HOZZAIRAS
185 REM ---------
190 OPEN 2,8,2,"PROBA1,S,A" : PRINT "A";
195 FOR I=4 TO 6
200 PRINT#2,I : PRINT TAB(2);I
205 NEXT I
210 CLOSE2
215 :
220 REM OLVASAS
225 REM -------
230 OPEN 2,8,2,"PROBA1,S,R": PRINT"S"; TAB(8);"R";
235 INPUT#2,X : PRINT TAB(10);X
240 IF ST=0 THEN GOTO 235
245 CLOSE2
```

Szekvenciális file-ok lezárása
Szekvenciális file-okat a CLOSE lf utasítás segítségével kell lezárni, ahol lf a szóban forgó file logikai file száma.

Szekvenciális file-ok írása/olvasása

A szekvenciális file-okat a PRINT#,INPUT#,GET# utasításokkal használhatjuk. Az egyes utasítások alakja a következő:

PRINT#lf [,<nyomtatási kép>]

Az utasítás a <nyomtatási kép>-ben szereplő értékeket a megadott formába kiírja az lf logikai file számu file-ba.

GET#lf,<változólista>

Az utasítás az lf logikai file számú file következő karaktereit beolvasva és a változólista elemeihez rendeli. Ismeretlen struktúrájú file-okat a legegyszerűbb a GET# utasítás segítségével olvasni.

INPUT#lf,<változólista>

Az utasítás beolvasva az lf logikai file számu file karaktereit az első kocsivissza-soremelés (kódja 13) karakterig. Az így kapott input-sor elemeit rendeli hozzá a <változólista> elemeihez úgy, mintha a billentyűzetről gépeltük volna be. A különböző mennyiségeket tehát elválasztó jelekkel (vessző, pontosvessző, kettőspont, kocsivissza-soremelés) kell elválasztani. Ha tehát különböző mennyiségeket INPUT#-al akarunk visszaolvasni, akkor a kiíráskor nekünk kell az elválasztó jelekről gondoskodnunk. Ha tehát egy file első három elemének az A$,B,C% változók értékét szeretnénk kiírni úgy, hogy az INPUT#-kal visszaolvasható legyen, akkor a következő két módon járhatunk el:

i/

```
10 OPEN 2,8,2,"@:PROBA,S,W"
20 INPUT A$,B,C%
30 PRINT# 2,A$;",";B;",";C%
40 CLOSE 2
```

Ebben az esetben a 30. sorban a program gondoskodik az elválasztó jel (,) kiírásáról.

ii/

```
10 OPEN 2,8,2,"@:PROBA,S,W"
20 INPUT A$,B,C%
30 PRINT# 2,A$
40 PRINT# 2,B
50 PRINT# 2,C%
60 CLOSE 2
```

Ebben az esetben a 30.-50. sorokban a PRINT# utasítások maguk gondoskodnak egy CHR(13) karakter kiírásáról.

Mindkét esetben a három értéket a következő programmal olvashatjuk vissza:

```
    100 OPEN 2,8,2,"PROBA,SEQ,READ"
    110 INPUT# 2,A$,B,C%
    120 PRINT A$,B,C%
    130 CLOSE 2
```

Az i/ használatakor a következő karakterek iródnak a PROBA nevű file-ba (A$="ILDIKO",B=2500,C%=3 esetén):

```
    ILDIKO, 2500 , 3 CHR(13)EOF
```

Ugyanez a ii/ esetben:

```
    ILDIKOCHR(13) 2500 CHR(13) 3 CHR(13)EOF
```

Az ST változó

A BASIC és a DOS külön is figyeli, hogy mikor érünk egy szekvenciális file végére. Ha egy szekvenciális file utolsó karakterét olvastuk be, az ST változó értéke 64 lesz. Igy lehetőség van olyan szekvenciális file olvasására is, amelyiknek nem ismerjük sem a hosszát, sem a szerkezetét. A következőkben egy olyan programot irunk, amelyik egy szekvenciális file-t karakterenként beolvas, és a karaktereket kiírja a képernyőre. Egyetlen módositást végez, a CHR$(13) karaktert inverz < jelre cseréli. (Vizsgáljuk meg programunkkal az előző programokkal kapott file-jainkat !)

```
100 REM *********************
105 REM * DSK.SEQLIST       *
110 REM *********************
115 REM * SEQ TIPUSU, ASCII *
120 REM * FILE-OK LISTAZASA *
125 REM *********************
130 :
135 INPUT "<CLR>FILE NEVE=";A$
140 OPEN 2,8,2,A$+",S,R"
145 :
150 IF ST<>0 THEN CLOSE2: END
155 GET#2,A$
160 IF A$=CHR$(13) THEN PRINT "<CTRL-RVSON><<CTRL-RVSOFF>";:GOTO
150
165 PRINT A$;
170 GOTO 150
```

5.5.2 Relatív file-ok használata

A relatív file-ok, hasonlóan a szekvenciális file-okhoz, rekordokból állnak, de ezek a rekordok ugyanolyan hosszúak. (éppen ezért nincs szükség a rekord végét jelölő speciális karakterek használatára.) Szemben azonban a szekvenciális file-okkal, ezekre a rekordokra sorszámuk alapján lehet hivatkozni. Ahhoz, hogy a DOS gyorsan megtaláljon egy-egy adott rekordot, külön blokkokra van szükség, amelyek megadják, hogy egy-egy rekord melyik blokkon található.

Relatív file-ok megnyitása/lezárása

Egy még nem létező relatív file-t a következő utasítással nyithatunk meg:

OPEN lf,<egységszám>,<csatorna>,"<név>,L,"+CHR$(<hossz>)

lf a logikai file szám, <név> a relatív file neve, <csatorna> a programban még eddig nem használt, 2-14 adatcsatornák bármelyike lehet. Az "L" betű a relatív file-t jelenti. <egységszám> a lemezes egység száma, általában 8. <hossz> a relatív file **rekordhosszát** adja meg, és legfeljebb 254 lehet. A file egész 'élete' alatt ez lesz a rekordhossz. Az utasítás fenti formájában hiába használjuk a "@:<név>" alakot, a file - ha esetleg létezett - nem fog törlődni. A relatív file-ok csak a SCRATCH DOS paranccsal törölhetők. Már meglevő relatív file-t az

OPEN lf,<egységszám>,<csatorna>,"<név>"

utasítással lehet megnyitni. A READ és WRITE megjelölésre nincs külön szükség; lévén, hogy a relatív file-ba írni, olvasni egyszerre lehet.

A megnyitott file-t a CLOSE lf utasítással zárhatjuk le.

A rekordszám beállítása

Relativ file-okat ugyanúgy írhatunk, olvashatuhk mint szekvenciális file-okat, azzal a lényeges különbséggel, hogy megadhatjuk hányadik rekord hányadik pozíciójától kezdődjék az I/O művelet. Erre szolgál a P DOS parancs, melynek alakja:

```
PRINT#hf,"P"+CHR$(<csatorna>)+CHR$(L)+CHR$(H)+CHR$(P)
```

hf a 15. csatorna megnyitásakor használt logikai file szám, <csatorna> a relatív file megnyitásában használt csatornaszám (secondary adress). L és H a rekord sorszámának alsó és felső byte-ja. A valódi sorszám tehát 256*H+L. Végül P a rekordon belüli kezdőpozíció.

Azt, hogy egy relatív file-nak hány rekordja van, a file 'élete' során a P parancsban kiadott legnagyobb rekordszám dönti el. Ha a file hosszánál nagyobb rekordszámra hivatkozik a P parancs, a DOS helyet foglal a lemezen a további rekordoknak. A még nem használt rekordok első byte-ja $FF, a többi 0. Ha egy rekordot nem írunk tele, a maradék rész 0-kal lesz feltöltve.

Relatív file-ok írása/olvasása

A GET#, INPUT# illetve PRINT# utasításokat használhatjuk a relatív file-ba való olvasásra és írásra. Az I/O művelet a legutolsó P DOS-parancsban megadott pozíciótól kezdi a file-t feldolgozni. GET# az azon a pozíción levő karaktert adja vissza, INPUT# a legközelebbi CHR$(13)-ig olvassa a file-t (akár a rekord végén túl is!), PRINT# az adott pozíciótól kezdve ír, de nem lépi át a rekordhatárt. A rekordba be nem férő karakterek elvesznek. Kivétel természetesen a file utolsó rekordja, amelyen ha túl olvasunk vagy túlírunk, hibajelzést kapunk (ST=64). Mivel azonban a rekord vége általában logikai határ is, célszerű elkerülni az olyan I/O műveletet, amelyik átlépi a rekord határát.

Példánkban egy számokat tartalmazó file-t nyitunk meg, és beleírunk 30 számot. A következő programrész segítségével tetszőleges számot visszaolvashatunk a lemezről és ha akarjuk, módosíthatjuk. Egy-egy számot a relatív file egy rekordja tartalmaz.

```
100 REM ****************************
102 REM *                          *
104 REM *                          *
110 REM * PELDA RELATIVE FILE-RA   *
120 REM *                          *
130 REM * A PROGRAM 30 SZAMOT IR   *
132 REM * EGY RELATIV FILE-BA.     *
134 REM * A SZAMOK AZUTAN TETSZES*
136 REM * LEKERDEZHETOK ES HA      *
138 REM * MODOSITHATOK.            *
140 REM *                          *
142 REM ****************************
150 :
```

```
153 REM A RELATIVE FILE MEGNYITASA
200 OPEN 15,8,15:REM  PRINT#15,"S:PROBA"
210 OPEN 1,8,2,"PROBA,L,"+CHR$(14)
212 :
214 REM 30 SZAM FELIRASA
215 REM A SZAMOK MEGEGYEZNEK A REKORD SORSZAMAVAL
216 REM HA MAGUNK AKARJUK MEGADNI OKET
217 REM A SORBAN ALLO MEGJEGYZESBEN
218 REM LEVO UTASITAST KELL HASZNALNI
220 FOR I=1 TO 30
230 SZAM=I:REM INPUT "KOV.SZAM";SZAM
232 :
234 REM A REKORDSZAM BEALLITASA
240 PRINT#15,"P"+CHR$(2)+CHR$(I)+CHR$(0)+CHR$(0)
242 REM A HIBA ELLENORZESE
250 INPUT#15,E1,E2$,E3,E4:IF(E1<20) OR E1=50 THEN GOTO 270
260 PRINT E1,E2$,E3,E4:CLOSE 1: CLOSE 15:END
262 :
263 REM A SZAM KIIRASA
270 PRINT#1,SZAM;
280 INPUT#15,E1,E2$,E3,E4:IF(E1<20) OR E1=50 THEN GOTO 300
290 GOTO 260
292 :
294 REM A CSATORNAK LEZARASA
300 NEXT: CLOSE 1: CLOSE 15
302 :
304 REM MODOSITO RESZ. HA A PROBA NEVU
305 REM FILE A 30 SZAMMAL MAR LETEZIK
306 REM INNEN KELL A PROGRAMOT INDITANI
310 OPEN 15,8,15: OPEN 1,8,2,"PROBA"
312 :
314 REM MODOSITO RESZ
320 INPUT"MELYIK SZAM KELL";I
330 I=INT(I):IF (I<0) OR (I>29) THEN GOTO 320
340 PRINT# 15,"P"+CHR$(2)+CHR$(I)+CHR$(0)+CHR$(0)
345 REM A SZAM BEOLVASAS
350 INPUT#1,SZAM
360 PRINT SZAM
370 PRINT "AKARJA-E MODOSITANI(I/N)"
380 GET A$: IFA$="" THEN GOTO 380
390 IF A$="N" THEN GOTO 430
400 INPUT "UJ SZAM ERTEKE";SZAM
410 PRINT#15,"P"+CHR$(2)+CHR$(I)+CHR$(0)+CHR$(0)
420 PRINT#1,SZAM
430 PRINT"ELEG VOLT-E (I/N)"
440 GET A$: IF A$="" THEN GOTO 440
450 IF A$="N" THEN GOTO 320
452 :
454 REM A CSATORNAK LEZARASA
460 CLOSE 1: CLOSE 15: END
```

5.6 Direkt elérési mód

A bevezetőben már említettük, hogy lehetőség nyílik az egyes blokkok közvetlen írására, illetve olvasására. Ahhoz, hogy a DOS ebben az üzemmódban dolgozzon egy # nevű pszeudo-file-t kell megnyitnunk, ami valójában a lemezegység valamelyik pufferét jelenti. (Ezenkívül mindenképp szükségünk van a 15. csatorna megnyitására is):

OPEN lf, <lemezegység>, <csatorna>, "#[<sorszám>]"

lf a file logikai file-száma, amire az I/O utasitásokban hivatkoznunk kell, <lemezegység> a lemezegység hardver száma (ez általában 8); <csatorna> azt az adatcsatornát jelenti, amin keresztül az adatátvitel zajlik. értékének a 2-14 tartományba kell esnie. Az opcionális <sorszám> a puffer sorszámát adja meg. Ha elmarad, a DOS maga választja meg, melyik puffert használja. Például

OPEN 15,8,15:OPEN 1,8,2,"#"

utasítja a DOS-t a közvetlen elérésű üzemmód használatára. A lemez és a puffer közti adatátvitelt a 15. csatornán DOS-parancsokkal szabályozhatjuk. A puffert az INPUT# és a GET# utasításokkal, mint 1-es file-t olvashatjuk, illetve a PRINT# utasítással írhatunk bele.

A következőkben összefoglaljuk a közvetlen elérésű üzemmód DOS parancsait.

B-R

BLOCK-READ utasítás

A DOS-parancs lehetővé teszi egy tetszőleges blokk olvasását.

Szintaxis:
PRINT#hf,"BLOCK-READ:";<csatorna>;<meghajtó>;<sáv>;<szektor>
vagy
PRINT#hf, "BLOCK-READ:<sztring>"

Az utasítás fenti alakjában a hf a 15. csatornának megfelelő logikai file-szám (ami általában 15). <csatorna> a # pszeudo--file megnyitásában szereplő csatornaszám. <meghajtó> jelen

esetben mindig 0. (Olyan lemezegység esetén, amelyik két lemezmeghajtót is tartalmaz, <meghajtó> értéke 0 vagy 1 lehet.) A <sáv> és <szektor> annak a blokknak a paraméterei, amit a pufferbe szeretnénk beolvasni. Az utasítás második formájában a <sztring> négy karakterének ASCII kódja felel meg a fenti négy értéknek. A "BLOCK-READ:" szövegrész az utasításban a "B-R:" és "U1:" szöveggel helyettesíthető. (Lásd a USER utasítást.)

Következő példánk a lemez megadott blokkját olvassa be a C-16 memóriájába és írja ki a képernyőre:

```
10 OPEN 15,8,15:OPEN 1,8,2,"#"
20 INPUT "<CLR>SAV,SZEKTOR";S,SZ
30 PRINT#15,"B-R:";2;0;S;SZ
40 GET#1,X$:IF ST=64 GOTO 20
50 PRINT X$;:GOTO 40
```

A szekvenciális file-okról szóló részben említettük, hogy a szekvenciális file-ok blokkjainak első 2 byte-ja a file következő blokkjának paramétereit adja meg. Az alábbi program lehetővé teszi az egymáshoz láncolt blokkok követését:

```
10 OPEN 15,8,15:OPEN 1,8,2,"#"
20 INPUT "<CLR>SAV,SZEKTOR";S,SZ
30 PRINT#15,"B-R:";2;0;S;SZ
35 PRINT#15,"B-P:";2;0
40 GET#1,X$:IF X$="" THEN X$=CHR$(0)
50 GET#1,Y$:IF Y$="" THEN Y$=CHR$(0)
55 S=ASC(X$):SZ=ASC(Y$)
60 PRINT "SAV=";S,"SZEKTOR=";SZ
65 GET A$:IF A$="" THEN GOTO 65
70 IF S=0 THEN CLOSE1:CLOSE15:END
75 GOTO 30
```

B—W

BLOCK-WRITE utasítás

Ez a DOS-parancs lehetővé teszi egy tetszőleges blokk írását.

Szintaxis: PRINT# hf, "BLOCK-WRITE:"+<sztring>

Az utasítás fenti alakjában hf a 15. csatornának megfelelő logikai file szám (ami általában ugyancsak 15). A csatorna a # pszeudo-file megnyitásában szereplő csatornaszám. A <meghajtó> jelen esetben mindig 0. (Olyan lemezegység esetében, amelyik két lemezmeghajtót is tartalmaz, a <meghajtó> értéke 0 vagy 1 lehet.) A <sáv> és <szektor> annak a blokknak a paraméterei, ahová a puffer tartalmát ki akarjuk írni. Az utasítás második

formájában a <sztring> négy karakterének ASCII kódja felel meg a fenti négy paraméternek. A "BLOCK-WRITE:" szövegrész az utasításban a "B-R:" és "U2:" szöveggel is helyettesíthető. (Lásd a USER utasítást.)

Első példánk az 1. sáv valamennyi (0-20 sorszámú) szektorába beírja ugyanazt az üzenetet. A szektorok sorrendjének változtatásával ki lehet próbálni, melyik esetben a leggyorsabb a DOS.

```
10 OPEN 2,8,2,"#":OPEN 15,8,15
20 DATA 0,1,2,3,4,5,6,7,8,9,10
30 DATA 11,12,13,14,15,16,17,18,19,20
40 INPUT "UZENET";M$
45 FOR I=1 TO 21:READ SZ
50 PRINT#15,"B-R:";2;0;1;SZ
60 PRINT#15,"B-P:";2;0
70 PRINT#2,M$
80 PRINT#15,"B-W:";2;0;1;SZ
85 NEXT I
90 CLOSE2:CLOSE15:END
```

(A programot csak üres, de megformázott lemezen próbáljuk ki, hogy meg ne sértsünk egyetlen programot se!)

Második példánk egy adott blokkot részben (adott pozíciótól kezdődően) módosít:

```
10 OPEN 1,8,2,"#":OPEN 15,8,15
20 INPUT "SAV,SZEKTOR";S,SZ
30 INPUT "KEZDOPOZICIO";P
40 INPUT "UZENET";M$
50 PRINT#15,"B-R:";2;0;S;SZ
60 PRINT#15,"B-P:";2;P
70 PRINT#1,M$;
80 PRINT#15,"B-W:";2;0;S;SZ
90 CLOSE1:CLOSE15:END
```

(A programot csak üres lemezen próbáljuk ki, hogy meg ne sértsünk egyetlen programot se!)

B—E

BLOCK-EXECUTE utasítás

Az utasítás hatása hasonló egy program betöltéséhez, majd futtatásához. A lemeznek az utasításban specifikált blokkja betöltődik a pufferba és a program végrehajtása a puffer elején folytatódik. Amikor a gépi kódú program egy RTS-hez ér, az a

BASIC programba való visszatérést eredményezi. Ritkán használjuk, mert ehhez a DOS ROM részletes ismeretére van szükség.

Szintaxis:

```
PRINT#hf, "BLOCK-EXECUTE:";<csatorna>;<meghajtó>;<sáv>;<szektor>
vagy
PRINT#hf, "BLOCK-EXECUTE:"+<sztring>
```

Az egyes paraméterek jelentése megegyezik a B-W és B-R parancsoknál leírtakkal. A "B-E:" rövidítés használata megengedett.

B-A

BLOCK-ALLOCATE utasítás

Az utasításban specifikált blokknak a BAM-ban megfelelő bit alacsony lesz, jelezve a DOS-nak, hogy azt más célokra már nem használhatja. Amennyiben a blokk már foglalt 65, NO BLOCK, S, SZ hibaüzenetet kapunk, ahol S, SZ a következő, *még* nem foglalt blokk sáv- és szektorszámát adja meg.

Szintaxis:

```
PRINT#hf, "BLOCK-ALLOCATE:";<meghajtó>;<sáv>;<szektor> vagy
PRINT#hf, "BLOCK-ALLOCATE:"+<sztring>
```

Az egyes paraméterek jelentése megegyezik a B-R illetve a B-W utasításban leírtakkal. (A <csatorna> megadására értelemszerűen nincs szükség.) A "B-A:" rövidítés használata megengedett.

Példánk egy szubrutint mutat be, amelyik a következő üres (nem foglalt) blokk sáv- és szektorszámával tér vissza. A 65-ös hiba figyelésén kivül még arról is gondoskodni kell, hogy a blokk - néhány kivételtől eltekintve - nem lehet a katalógus része. A program az S, SZ értékpárban adja vissza a következő szabad blokk paramétereit. E értéke a hibakódot tartalmazza. Ez 0, ha sikerült szabad blokkot találni, különben a felmerült DOS-hiba kódját tartalmazza:

```
1000 PRINT#15,"B-A:";0;S;SZ
1010 INPUT#15,E,E$,ES,EZ
1020 IF E=0 THEN RETURN
1030 IF E<>65 THEN RETURN
1040 S=ES:SZ=EZ:IF S=18 THEN S=19
1050 GOTO 1000
```

B-F

BLOCK-FREE utasítás

A DOS-parancs a B-A parancs ellentettje. A B-F parancsban specifikált blokknak a BAM-ban megfelelő bit magas lesz, jelezve, hogy a továbbiakban a DOS azt más célokra felhasználhatja.

Szintaxis: PRINT#hf, "BLOCK-FREE:";<meghajtó>;<sáv>;<szektor>

Az egyes paraméterek jelentése megegyezik a B-R illetve a B-W utasításban leírtakkal. (A <csatorna> megadására értelemszerűen nincs szükség.) A "B-F:" rövidítés megengedett. Például a

PRINT#15,"B-F:";0;1;SZ

lemez 1. sávjának SZ-ik szektorát a DOS rendelkezésére bocsátja. (Csak akkor próbáljuk ki, ha tudjuk, hogy nem sértünk meg egyetlen programot sem!)

B-P

BUFFER-POINTER utasítás

A # pszeudo-file-hoz egy mutató is tartozik, amelyik a lehetséges (1-255-ig számozott) byte valamelyikére mutat. Az író/olvasó utasítások végrehajtása ettől a byte-tól kezdődik. A mutatót a B-P utasítás segítségével tetszőleges helyre pozícionálhatjuk. A mutató utolsó értéke, mint a blokk 0. byte-ja a lemezre íródik.

Szintaxis: PRINT#hf, "BUFFER-POINTER:";<csatorna>;<mutató>

A hf és a <csatorna> jelentése ugyanaz, mint a B-R utasításban; <mutató> a # pszeudo-file pufferének pointerét állítja át. A "B-P:" rövidítés itt is megengedett. A B-P parancsot elsősorban a B-W és B-R utasításokkal együtt használjuk. (B-W példájában már szerepelt.)

M-R

MEMORY-READ utasítás

Az utasítás lehetővé teszi a lemezegység memóriájának byte-onkénti olvasását. (Akár a RAM-ból, akár a ROM-ból

olvashatunk.) Ez lehetővé teszi a DOS működésének tanulmányozását, a 0. lap megismerését stb. Minden egyes byte-ra egy új utasítást kell kiadni.

Szintaxis: PRINT#hf, "M-R:";CHR$(L);CHR$(H)

Az utasítás fenti alakjában hf a lemezegység 15. csatornájához tartozó logikai file szám; L és H a memória címének alsó illetve felső byte-ja. Az utasítást követően a szóban forgó memória címen levő byte a GET#hf, A$ utasítás segítségével olvasható, ahol hf a hiba-csatornának megfelelő logikai file szám.

M—E

MEMORY-EXECUTE utasítás

Az utasítás lehetővé teszi a lemezegység tetszőleges címén kezdődő (gépi kódú) program végrehajtását. Ez lehet egy ROM alprogram, de lehet a RAM-ban az M-W parancs segítségével megírt program is. A szintaxis megegyezik az M-R utasítás szintaxisával.

Szintaxis: PRINT#hf, "M-E:";CHR$(L);CHR$(H)

ahol L és H a kezdőcím alsó illetve felső byte-ja.

M—W

MEMORY-WRITE utasítás

A parancs lehetővé teszi - egyidőben 34 byte - beírását a DOS memóriájába. Ezeket azután például az M-E parancs használhatja.

Szintaxis:
PRINT#hf, "M-W:";CHR$(L);CHR$(H);<karakterek száma>;<byte-ok>

hf, L és H jelentése ugyanaz, mint az M-R utasításban. Ezt követi a <karakterek száma> paraméter, amelyik definiálja, hány byte-ból áll a DOS memóriájába írandó sorozat. Ezt követik a sorozat byte-jai CHR$(B) alakban. Például

```
PRINT#15, "M-W:";CHR$(0);CHR$(5);1;CHR$(96)
```

a $0500 címre egyetlen byte-ot (96) helyez el. Ez egy RTS utasításnak felel meg.

USER

Ez a DOS-parancs lehetővé teszi a DOS memóriájában tárolt bizonyos címekre való ugrást. Ezek általában további JMP utasításokat használnak, amelyek lehetővé teszik egy tetszőleges DOS rutin elérését.

Szintaxis: PRINT#hf, "<USER utasítás>"

hf a 15. csatorna megnyitásában használt logikai file szám (általában maga is 15). Az utasításhoz néha további byte-okat is el kell küldeni. A <USER utasítás> lehetséges értékeit és azok jelentését a következő táblázat foglalja össze:

USER utasítás	*Jelentés*
U1 vagy UA	B-R a puffer-pointer felhasználása nélkül
U2 vagy UB	B-W a puffer-pointer felhasználásával
U3 vagy UC	JMP $0500
U4 vagy UD	JMP $0503
U5 vagy UE	JMP $0506
U6 vagy UF	JMP $0509
U7 vagy UG	JMP $050C
U8 vagy UH	JMP $050F
U9 vagy UI	JMP $FFFA (=NMI vektor)
U; vagy UJ	RESET vektor
UI+	C-64 sebességének kiválasztása
UI-	C-16 sebességének kiválasztása

Külön szólunk az U1 és U2 hatásáról. A B-R utasítás az adott blokkot *csak* a blokk első byte-jaként tárolt puffer-mutató értékéig olvassa be. Ha az utasítás U1 alakját használjuk, akkor mind a 256 byte a pufferbe kerül. A B-W utasítás a puffer-pointer értékét és mind a 255 adat-byte-ot az adott blokkba kiírja. Az utasítás U2 alakja a puffert csak a lemezen levő mutató értékéig másolja vissza.

6. Fejezet

Grafikus lehetőségek. Hanggenerálás

6.1 Bevezetés

A C-16 mikroszámítógép elsődleges kimeneti eszközként (outputként) a televíziós képernyőt (VDU, video display unit) használja. Ez azt jelenti, hogy az interpreter üzenetei, a PRINT utasitás segitségével kiíratott értékek, az INPUT-promt mind az elsődleges outputon jelennek meg. A CMD utasítás segítségével megváltoztathatjuk az elsődleges outputot. A számitógép bekapcsolásakor az elsődleges output azonban a képernyő. **A képernyő egy homogén, egyszínü *keretből* és az igazi információt hordozó, 320x200 pontból álló *pontrácsból* áll.** A pontrács pontjai - bizonyos korlátozással - a C-16 által használt 121-féle színnel lehetnek kiszínezve. (Ha csak kétféle színt használunk, akkor az egyes pontokról azt mondjuk, hogy be-, illetve kikapcsolt állapotban vannak.) A 320x200-as pontrács a karakter üzemmódok esetén 8x8-as részekre esik szét. Igy összesen 25 sor és 40 oszlop keletkezik. Ezeket a 8x8-as részeket - mivel éppen egy karakter tárolására alkalamasak - **karakterhelyeknek** hívjuk.

A C-16 számítógépen a következő grafikus üzemmódokat használhatjuk:

A./ Karakteres üzemmódok
1./ Standard karakteres üzemmód
2./ Bővített háttérszín üzemmód

B./ Bittérképes üzemmódok
1./ Standard bitttérképes üzemmód
2./ Többszínü bittérképes üzemmód
3./ 'Vágott' képernyő

Mindezeket a funkciókat a TED jelű video chip biztositja, amely a C-16 BASIC utasításai segítségével programozható. Ez lényeges könnyebbség a C-64 vagy a VC-20 számítógépekhez képest, ahol erre a célra csak a PEEK/POKE utasításpárt használhattuk. A video chip regisztereinek tartalmát az interpreter mindig megfelelően állítja be. Ezen regiszterek azonban csak a kép

előállításának módját határozzák meg. A memória bizonyos területeinek a tartalma határozza meg, hogy mi jelenjen meg a képernyőn. Ezek a memóriaterületek a karakteres, illetve a bittérképes üzemmódban eltérnek egymástól.

6.2 Karakteres üzemmód

Ezt a fajta üzemmódot alacsony felbontású üzemmódnak is szokás nevezni. Ebben az esetben a képernyő egy karakternyi (másként 8x8 pontnyi) helyét a karakter memóriában meghatározott 256 karakter valamelyike töltheti be. A képernyő 25 sorra és 40 oszlopra tagozódik,és a szerkesztő karakterek segítségével könnyen beállíthatjuk azt a poziciót, amelyikbe írni szeretnénk. A karakter üzemmódok csak annyiban térnek el egymástól, hogy a karakterek színe és alakja másként jön létre a képernyő -,a szín- és a karakter memóriából.

Képernyő memória
A képernyő memória a képernyőre kiírandó karakterek kódját tartalmazza. Ennek megfelelően a képernyő memória 40*25=1000 byte helyet foglal el. Elhelyezése: 3072-4071 ($0C00-$0DBE)

Színmemória
A színmemória a képernyőre kerülő karakterek színét, fényerejét, továbbá villogását határozza meg. A színmemória a 2048-3047 ($0800-$BBE) címeken található meg. Az alsó négy bit a karakter színét, a 4.-6. bitek a fényerejét határozzák meg. Ha a színmemóriában a 7. bit magas (=1), akkor a megfelelő karakter villogni fog. Ha alacsony (=0), akkor normál módban kerül kijelzésre.

Karakter memória
Bármelyik grafikus jel egy 8x8-as pontrács segítségével ábrázolható, a pontok mindegyike bekapcsolt (1) vagy kikapcsolt (0) állapotban van. *A C-16 a karakterek alakját a karakter generátor ROM-ban tárolja.*

Egy karakter alakjának tárolásához 8 byte-ra van szükség. Minden egyes byte a karakter valamely sorát (azon belül minden bit egy pontját) reprezentálja. A 0 bit azt jelenti, hogy a pont ki van kapcsolva, az 1-es bit pedig azt, hogy a pont be van kapcsolva.

A karakter memória a ROM-ban az 53248 címen kezdődik. Az első 8 byte az 53248 ($D000) helytől az 53255 helyig ($D007) a @ jel alakját tartalmazza, amelynek a képernyő kódja 0. A következő 8 byte, az 53256 ($D008) helytől az 53263 ($D00F) helyig az A betű pontmátrixát tartalmazza:

Kép	Bináris	Deci-mális	Cim
**	00011000	24	53256
****	00111100	60	53257
** **	01100110	102	53258
******	01111110	126	53259
** **	01100110	102	53260
** **	01100110	102	53261
** **	01100110	102	53262
	00000000	0	53263

Az egyenként 256 karakterből álló karakterkészletek a memóriában külön-külön 2K (2048 byte) helyet foglalnak, karakterenként 8 byte-ot. Mivel összesen két karakterkészlet van, ezért a karakter generátor ROM 4K helyet foglal el.

Bakapcsoláskor a C-16 standard karakteres üzemmódban dolgozik. *Ha ettől eltérő módból szeretnénk visszatérni, akkor* a ***GRAPHIC O BASIC*** *utasítást kell végrehajtanunk.* Ebben az üzemmódban a keret, a háttér (papír), illetve az írás (tinta) színét állíthatjuk be a következő utasításokkal:

COLOR 0,<szín>,<fényerő> (háttér színe)
COLOR 1,<szín>,<fényerő> (írás színe)
COLOR 4,<szín>,<fényerő> (keret színe)

A képernyőre a PRINT és a CHAR utasítás segitségével írhatunk. Az utóbbiban az írás kezdőpozícióját is megadhatjuk.

Bővitett háttérszín üzemmód
A bővitett háttérszín üzemmód lehetőséget nyújt arra, hogy minden egyes karakter **háttér színét** külön is beállítsuk. Például ebben az üzemmódban megjelenithetünk egy kék karaktert sárga háttérrel egy fehér képernyőn.

A bővített háttérszín üzemmódnak négy regiszter áll rendelkezésére. Mindegyik regiszter a 121 szín bármelyikére állítható. Bővitett háttérszín üzemmódban a szín-memóriát az írás színének tárolására használjuk. A színmemória használata tehát ugyanolyan, mint a standard karakter üzemmódban.

A bővített háttérszín üzemmód határt szab a különböző megjeleníthető karakterek számának. Ha a bővített háttérszín üzemmódot használjuk, akkor csak a karakter ROM-ban levő első 64 karaktert jeleníthetjük meg. Ez azért van, mert ilyenkor a video chip a karakter képernyő kódjának első két bitjét a háttérszín kódjának meghatározására használja fel.

Az 'A' betű kódja 1. Ha a bővített háttérszín üzemmódot használjuk, és a POKE utasítással 1-et viszünk a képernyő memóriába, az 'A' betű fog megjelenni. Ha POKE utasítással 65-öt viszünk a képernyőre, akkor azt várnánk, hogy a 129-es karakter kóddal rendelkező karakter jelenik meg, ami az inverz A. Nem ez történik. A bővített háttérszín üzemmódban ehelyett ugyanazt a nem inverz 'A' betűt kapjuk mint az előbb, de más háttérszínnel. A következő táblázat megadja az összefüggést:

Karakter kód intervallum	**bit 7**	**bit 6**	**Háttérszín szám**	**regiszter cím**
0-63	0	0	0	65301 ($FF15)
64-127	0	1	1	65302 ($FF16)
128-191	1	0	2	65303 ($FF17)
192-255	1	1	3	65304 ($FF18)

A regiszterek alsó négy bitje a színt, a 4.-6. bitek pedig a fényerőt határozzák meg.

A bővített háttérszín üzemmód használatához a TED $FF06 című regiszterének 6. bitjét 1-re kell állítani. Ezt a következő POKE utasítással érhetjük el:

```
V=DEC("FF06"): POKE V, PEEK(V) OR 64
```

A bővített háttérszín üzemmód ugyanennek a bitnek 0-ra állításával kapcsolható ki. Ezt a következő utasítás végzi el:

```
V=DEC("FF06"): POKE V, PEEK(V) AND 191
```

A következő program a bővített háttérszínű üzemmód hatását mutatja be. Bár a begépeléskor a kis és a nagybetűket egyaránt használtuk, a képernyőn ugyanaz a nagybetűs felirat jelenik meg, négy különböző háttérszínnel:

```
10 v=dec("ff06")
20 poke v,peek(v) or 64
30 print"<CLEAR>i love commodore"
40 print"I LOVE COMMODORE"
50 print"<RVS ON>i love commodore"
60 print"<RVS ON>I LOVE COMMODORE"
65 getkey a$
70 poke v,peek(v) and 191
```

6.3 Bittérképes üzemmód

A video chip másik alapvető üzemmódja az, amikor a 320x200-as képernyő minden egyes pontját külön-külön ki-, vagy bekapcsolhatjuk a memória megfelelő bitjének alacsonyra (0) vagy magasra (1) állításával. A bittérképes üzemmód hátránya, hogy közel 8 Kbyte a helyfoglalása a memóriában (=320x200/8). A bittérképes üzemmódot éppen ezért nagyfelbontású üzemmódnak is szokás hívni, hiszen karakteres üzemmódban csak a rögzített (vagy a magunk által programozott) grafikus jeleket használhatjuk.

A bittérkép használata nagy mértékben csökkenti a BASIC program rendelkezésére álló munkaterületet. Ha már nincs rá szükség, akkor a GRAPHIC CLR utasítás felszabadítja ezt a helyét. A GRAPHIC 0 utasítás erre nem alkalmas. Igaz, hogy nem látjuk a nagyfelbontású képernyőt, de azért az még foglalja a helyet. Erről azonnal meggyőződhetünk, ha a GRAPHIC 1 parancsot kiadjuk. (Újra látjuk az előző képet!)

A Commodore 16-on a bittérképezés két fajtáját használhatjuk:

1. **Standard (*nagyfelbontású*) bittérképes** üzemmód
   (320x200 pontos felbontás)
2. ***Többszínű*** **bittérképes** üzemmód
   (160x200 pontos felbontás).

Az elsőre a GRAPHIC 1, a másodikra a GRAPHIC 2 utasítás kiadásával térhetünk át. A kétfajta üzemmód a bittérkép szinezésével függ össze. Ha azt szeretnénk, hogy a képernyő minden egyes raszterpontja a lehetséges 121 szín valamelyike lehessen, akkor nem 8Kbyte, hanem kb. 480 Kbyte memóriára lenne szükség. Ezen a számítógépek konstruktőrei különböző trükkökkel igyekeznek segíteni. Standard bittérképes üzemmód esetén a bittérkép egy karakterhelyre eső pontjai két féle színűek lehetnek, s hogy melyik az a két szín, azt a szín-, és fényerő memória határozza meg. A bittérkép bitjeinek színe tehát így alakul:

Bit	**Szín és fényerő**
0	A megfelelő memória alsó négy bitje
1	A megfelelő memória felső négy bitje

A színeket a COLOR utasítás segítségével állíthatjuk be. Vigyázzunk, az egyes színek hatása csak akkor érződik, ha rajzolunk. Jól látszik ez a következő program végrehajtása közben:

```
10 GRAPHIC 1,1
20 COLOR 1,15,0
30 COLOR 0,14,5
40 COLOR 4,7,0
50 FOR X=0 TO 2*<PI> STEP 2*<PI>/30
60 CIRCLE 1,45*SIN(X)+159,45*COS(X)+100,45,45
70 NEXT X
80 GETKEY A$: GRAPHIC 0
```

Az egyes memóriarészek a következőképpen helyezkednek el:

Memória	**Funkció**
6144- 7167 ($1800-$1BFF)	Fényerő
7168- 8191 ($1C00-$1FFF)	Színmemória
8192-16383 ($2000-$3FFF)	Bittérkép

Standard bittérképes üzemmód
A standard bittérképes üzemmód 320 vizszíntes, 200 függőleges pontú felbontást ad, minden egyes 8x8-as karakterhelyen két szín közötti választás lehetőségével.

A bittérkép használata
A pontok ki-, illetve bekapcsolásához tudnunk kell, hogy egy adott képernyőponthoz hogyan lehet megtalálni a megfelelő bitet a memóriában. Ennek kiszámításához a 320x200-as pontrácsot a következő koordináta rendszerben helyezzük el:

```
0--------------------X-----------------319
.                                      .
.                                      .
.                                      .
.                                      .
Y                                      .
.                                      .
.                                      .
.                                      .
.                                      .
199------------------------------------.
```

A pont függőleges és vizszíntes helyzetének meghatározásához az X és Y betűket fogjuk használni. Az X=0 és Y=0 pont tehát a képernyő bal felső sarkában van. Minél nagyobb az X érték, a pont annál inkább jobbra, és minél nagyobb az Y értéke, annál inkább lejjebb van.

Az egyes pontoknak megfelelő bitek sajnos nem sorfolytonosan vannak a memóriában elhelyezve, hanem karakterhelyenként. Ezen belül pedig sorfolytonosan (úgy, ahogy az egyes karaktereket definiálhatjuk). Igy a bit-térkép 2. byte-ja a képernyő 2. sora első 8 pontjának állapotát határozza meg. (A pontok koordinátái: (0,1)...,(7,1).)

Az X, Y koordinátáju pont ki-, vagy bekapcsolásához pontosan tudni kell, hogy a bittérkép melyik byte-jának melyik bitje felel meg az (X,Y) pontnak. Ehhez először meghatározzuk, melyik karakterhelyen áll a pont:

```
OSZLOP = INT(X/8) : SOR = INT(Y/8)
```

Következő lépés, hogy meghatározzuk, hogy az adott karakterhely hányadik sorában áll:

```
SRKAR=Y-INT(Y/8)*8
     =Y AND 7
```

Az oszlopszám:

```
OKAR =X-INT(X/8)*8
     =X AND 7
```

Ez alapján a byte és a bit már adódik:

```
BIT =7-OKAR  ( =7-X AND 7)
BYTE = 320*SOR + 8*OSZLOP + SRKAR + DEC("2000")
```

Az (X,Y) pont bekapcsolása a következő utasitással történhet:

```
POKE BYTE,PEEK(BYTE) OR 2^BIT
```

Hasonlóan történik a pont kikapcsolása is:

```
POKE BYTE,PEEK(BYTE) AND (255-2^BIT)
```

Szerencsénkre a C-16 BASIC interpreter lehetőséget biztosit pontok, szakaszok, ellipszisek, illetve téglalapok rajzolására, igy a fenti hosszadalmas procedurát a valóságban a legritkább esetben kell csak elvégezni.

A TED bizonyos regiszterei határozzák meg a kijelzés módját. Ezek megfelelő beállitásával tér át az interpreter az egyik kijelzési módból a másikba. Ha tudjuk, hogy az egyes regiszterek tartalmának mi a jelentése, akkor a GRAPHIC utasitás nélkül is áttérhetünk egyik üzemmódról a másikra. (A C-64-en **csak** ezt lehet csinálni!) A következő kis példa ezt mutatja be:

```
10 GRAPHIC1,1
20 CIRCLE 1,100,100,80,80
30 GRAPHIC0
33 PRINT"<CLEAR><CRSR LE>  POKE-KAL VISSA!"
35 FOR I=1 TO 255: NEXT I
40 V=DEC("FF00")
50 POKE V+20,DEC("1F")
60 POKE V+18,DEC("C8")
70 POKE V+6,DEC("3B")
80 GETKEY A$
90 POKE V+20,DEC("0F")
100 POKE V+18,DEC("C4")
110 POKE V+6,DEC("1B")
```

Többszínű bittérképes üzemmód

A többszínű üzemmód lehetőséget biztosít arra, hogy egy karakterhelyre eső pontok négy féle színűek lehessenek. Ezek kézül kettő a szín- és fényerő memória által meghatározott szín, míg a másik kettő valamennyi karakterhelyre ugyanaz. Ebben az üzemmódban a bittérkép bitje-it a video chip párosával kezeli, s így összesen 4 bitkombináció jöhet létre. Ez lesz a négy előbb említett szín. Ez azonban azzal jár, hogy a szín meghatározásában résztvevő biteknek megfelelő két pont a képernyőn egyforma színű lesz. Többszínű üzemmódban így a vízszintes felbontás a felére csökken:

```
0--------------------X--------------------159
.                                         .
.                                         .
.                                         .
.                                         .
Y                                         .
.                                         .
.                                         .
.                                         .
.                                         .
199---------------------------------------.
```

Az egyes bitkombinációknak megfelelő színek a következők:

Bitek	Szin
00	Háttérszín#0
01	A szín- és fényerő memória felső négy bitje
10	A szín- és fényerő memória alsó négy bitje
11	Háttérszín#1

'Vágott' képernyő

A C-16 lehetőséget biztosít a karakteres és a bittérképes kijelzési mód egyidejű megjelenítésére. A képernyőt ebben az üzemmódban a video chip kettévágja: a karakteres képernyő utolsó 5, a nagyfelbontású képernyő első 20 (=160 raszter) sora látható. A képernyők maradék részének tartalma sem vész el, csupán nem látható. Erre az üzemmódra a

```
GRAPHIC 3   (az első 20 sor standard nagyfelbontású)
GRAPHIC 4   (az első 20 sor többszínű nagyfelbontású)
```

utasítások kiadásával térhetünk át.

6.4 További grafikus lehetőségek

A képernyő görgetése

A fenti alcím a 'scroll'ozással azonos tartalmat takar: a képernyőt valamilyen irányban soronként (oszloponként) eltoljuk. Soron itt nem egy karaktersor, hanem egyetlen rasztersor értendő. Ezzel az eljárással tetszőleges információt egyenletesen vihetünk fel a képernyőre. A képernyő görgetését a video-chip azon tulajdonsága teszi lehetővé, hogy a képernyő tartalma 0-15 raszter sorral (0-15 raszter oszloppal) eltolva jeleníthető meg.

A video chip $FF06-os regiszterének alsó négy bitje a függőleges, míg a $FF07 regiszterének alsó négy bitje a vízszintes eltolás mértékét határozza meg.

A képernyő kikapcsolása

A video chip $FF06 című regiszterének 4. bitje vezérli a video chip működését. Amikor ez a bit magas (1), a video chip normálisan működik. Ha ezt a bitet 0-ra állítjuk, a TED felfüggeszti működését, a képernyő a keret színére változik (és üres lesz). Az információ nem vész el, csupán nincs kijelezve. Ezt a hatást tapasztalhatjuk például a kazettát használó utasítások végrehajtásánál. A képernyő 'elsötétítését', illetve visszakapcsolását a következő utasítással érhetjük el:

```
V=DEC("FF06"): POKE V,PEEK(V) AND 239: REM SOTET
V=DEC("FF06"): POKE V,PEEK(V) OR 16: REM NORMAL
```

A képernyő elsötétítése valamelyest felgyorsítja a processzor működését, és így a BASIC programok végrehajtását is. Hosszabb számítások idejére célszerű elsötétíteni a képernyőt.

6.5 Hanggenerálás

A C-16 számítógépbe épített TED chip 3 független hang megszólaltatását teszi lehetővé. Ezek közül 2 normál zenei hang, míg a harmadik egy ún. fehér zaj. A hanggenerátor működését 5 regiszter vezérli. A $FF0D, $FF0E és a $FF0F regiszterek tartalma, továbbá a $FF10 regiszter 6.-7. bitjei együtt határozzák meg a megszólaló hang magasságát. Ez egyben azt is jelenti, hogy a három hang magasságának egymáshoz való viszonya nem lehet tetszőleges. A hangok magasságát meghatározó 6 bites számokból a felső két bit azonos. A szám és a neki megfelelő hangmagasság közti összefüggés az alábbi:

S=1024-(111840.45/FR)

ahol FR a hang frekvenciája Hz-ben, S pedig a szóbanforgó 6 bites szám.

A hang megszólalását a $FF11 regiszter tartalma határozza meg. Az egyes bitek jelentése a következő:

Bitek	Funkció
0.-3.	Hangerő
4.	Az 1. hang bekapcsolása
5.	A 2. hang bekapcsolása
6.	A 3. hang bekapcsolása

A hanggenerátor úgy működik, hogy a 4.-6. bitek valamelyikének magasra állítása a hang megszólalását, míg 0-ra állítása a kikapcsolását eredményezi. A hang a hangerő értékének megfelelő hangerővel szólal meg. Ha a hangerő 0, akkor hiába kapcsoljuk be, nem fogjuk hallani.

A C-16 BASIC V3.5 két utasitást biztosit a hanggenerátor programozására. A hangerőt a

VOLUME <hangerő>

utasítás kiadásával állíthatjuk be (A fentiekkel összhangban a <hangerő> értékének a 0-7 intervallumba kell esnie.) Ekkor még egyetlen hang sem szólal meg. Erre a SOUND utasítás szolgál, melynek alakja:

SOUND <hang>,<magasság>,<idő>

A <hang> értéke 0,1 vagy 2 lehet és ez az 1., a 2., illetve a 3. oszcillátort kapcsolja be. A <magasság> az előbbiekben már jelzett 6 bites szám, értéke így a 0-1023 intervallumba eshet.

Az <idő> a hang megszólalásának idejét határozza meg. Maximális értéke 65535 lehet, ez 1311 másodpercnek felel meg.

A SOUND végrehajtását a megfelelő hanghoz tartozó hangmagasság regiszter beállításával kezdődik, majd a $FF11 regiszter 4.,5., vagy 6. bitjének magasra állításával folytatódik. Az <idő> paraméternek megfelelő idő kivárása után az interpreter az előző bitet 0-ra állítja, s ezzel kikapcsolja a hangot.

Ezzel a két egyszerű utasítással is igen sokfajta zenei programot készithetünk. Igazán érdekes hatásokat azonban gépi kódú programokkal érhetünk el, azok ugyanis lehetővé teszik a hanggenerátor regisztereinek *igen gyors* változtatását is.

Telefon

Első példaprogramunk azt a hangot kíséreli meg visszaadni, amit akkor hallunk, amikor kicseng a telefon. Az 50-90 ciklusban először bekapcsolunk egy hangot (a 60. sorban), majd egy rövid idejű várakozás után (70-80. sorok) újból visszakapcsoljuk.

```
10 REM ***********
20 REM * TELEFON *
30 REM ***********
35 :
40 VOL 4
45 :
50 FOR L=1 TO 15
60 SOUND 1,770,45
65 :
70 FOR M=1 TO 900
80 NEXT M
85 IF L=13 THEN PRINT"<CLEAR>VEDD MAR FEEEEEL!!!!"
90 NEXT L
100 VOL 0
```

Urhajo

A program egy induló űrhajó hangját szimulálja. Az 50-80 ciklusban a megszólaló két hang magasságát folyamatosan növeljük. Mély dübörgésből így a végén éles süvítés lesz.

```
10 REM *******************
20 REM * INDUL AZ URHAJO *
30 REM *******************
40 VOL 7
50 FOR I=1 TO 1023 STEP 3
60 SOUND 1,I+1,1
70 SOUND 2,I+1,1
80 NEXT I
```

Sziréna

A program egy közeledő mentőautó hangját utánozza. A közeledés érzettét a hangerő folyamatos növelésével érjük el. Erre szzolgál 40-90 ciklus. A szirénahatást egy magas és egy mélyebb hang gyors egymásutánban történő megszólaltatásával érjük el.

```
10 REM ***********
20 REM * SZIRENA *
30 REM ***********
35 :
40 FOR I=1 TO 8
45 :
50 VOL I
60 FOR J=1 TO 5
70 SOUND 1,897,23
80 SOUND 1,685,23
90 NEXT J,I
100 :
110 PRINT"<CLEAR><RVS ON>ITT VAGYUNK! HOL A BETEG?"
```

Zongora

Utolsó példánk lehetővé teszi, hogy zongorázzunk a C-16-tal. A hatás persze nem teljesen azonos. Ha a zongorán például leütünk egyszerre 5 billentyűt, mind az öt hang egyszerre megszólal, a programunkban csak egymás után, abban a sorrendben, ahogy a billentyűzetet olvasó rutin érzékelte ezek lenyomását.

```
100 REM ***********
105 REM * ZONGORA *
110 REM ***********
115 :
120 REM KLAVIATURA KIRAJZOLASA
125 PRINT"▮▮▮▮▮▮▮"
130 PRINT"          R ▮ ▮ | ▮ ▮ ▮ | ▮ ▮ | ▮ ▮ "
135 PRINT"          R ▮ ▮ | ▮ ▮ ▮ | ▮ ▮ | ▮ ▮ "
140 PRINT"          R ▮ ▮ | ▮ ▮ ▮ | ▮ ▮ | ▮ ▮ "
145 PRINT"          R | | | | | | | | | | | | "
150 PRINT"          RQ|W|E|R|T|Y|U|I|O|P|§|+|-"
170 :
175 REM FREKVENCIAK BETOLTESE
180 DIMF(14):DIMK(255)
185 FORI=1TO14: READ F(I): NEXT I
195 :
200 REM KEZDOERTEK
205 K$="QWERTYUIOP§+-▮"
210 FORI=1TOLEN(K$):K(ASC(MID$(K$,I)))=I:NEXT
215 IOSEL 7
220 :
250 REM A BILLENTYUZET OLVASASA
255 GET▮ A$
260 FR=K(ASC(A$))
265 IF FR=0 THEN GOTO 255
270 :
275 REM EGY HANG MEGSZOLALTATASA
300 DIRECTORY 2,F(FR),6
305 GOTO255
306 DATA 169,262,345,383,453
310 DATA 515,571,596,643,685,704,739
320 DATA 770,798
```

7. fejezet

BASIC programozási példák

7.1 BASIC programstrukturák

Ebben a részben rövid példaprogramokat mutatunk be, amelyek lehetővé teszik a 3. fejezetben mondottak gyakorlását. A *BAS* előszóval kezdődő programok a BASIC programstruktúrák, illetve a ki- és beviteli utasitások gyakorlását szolgálják.

BAS.COSINUS

A program a 2. fejezetben szereplő "derékszögű háromszög" program "továbbfejlesztett" változata. Nem a Pitagorasz-tétel alapján számol, és így lehetőség nyílik nem csak derékszögű háromszögek harmadik oldalának kiszámítására is. Ebben az esetben természetesen meg kell adnunk a két ismert oldal által bezárt szöget is. Ennek megfelelően az INPUT utasítás segítségével az A,B,GAMMA értékeket olvassuk be. A C-16 interpreter a változóneveket első két karakterük alapján azonosítja, ezért GAMMA helyett elég a GA használata. A harmadik (C) oldalt a koszinusztétel segítségével számítjuk ki.

```
100 REM *****************
105 REM * BAS.COSINUS    *
110 REM *****************
115 REM * KOSZINUSZTETEL *
120 REM *****************
125 :
130 INPUT "<CLEAR>A,B,GAMMA=";A,B,GA
135 :
140 PRINT "C=";
145 PRINT SQR(A*A+B*B-2*A*B*COS(GA*<pi>/180))
150 END
```

BAS.FAKTOR

A programok az N! kifejezés értékét számítják ki. N! az első N szám szorzata. A három program alapelve ugyanaz, csak a megvalósítás tér el. Az S változóban tároljuk a részlet-szorzatokat, I pedig azt mutatja, hogy melyik számmal kell a szorzást folytatni. Az első két változatban a számokat alulról felfelé haladva szorozzuk össze, S értéke tehát rendre 1!, 2!,

3! stb. A BAS.FAKTORV1 és a BAS.FAKTORV2 programok közt csak annyi a különbség, hogy az előbbiben az **IF** feltételes vezérlésátadó utasítást használjuk, míg az utóbbiban a **FOR** ciklusutasítást. Látható, hogy a ciklusutasítás megkönnyíti ugyan a program megírását, de a program strukturája nem változik.

A BAS.FAKTORV3 program ugyancsak a **FOR** utasítást használja, de a számokat felülről lefelé szorozza össze. Ilyenmódon a ciklusutasításban szerepelnie kell a **STEP -1** lépésköz beállításnak is. S-ben igy rendre az

```
S=N;
S=N*(N-1);
S=N*(N-1)*(N-2) stb.
```

értékek kerülnek.

```
100 rem *****************
105 REM * BAS.FAKTORV1    *
110 REM *****************
115 REM * FAKTORIALIS V1 *
120 REM *****************
125 :
130 INPUT "<CLEAR>N=";N
135 S=1: I=1
140 :
145 S=S*I
150 I=I+1: IF I<=N THEN GOTO 145
155 :
160 PRINT "N!=";S
165 END
```

```
100 REM *****************
105 REM * BAS.FAKTORV2    *
110 REM *****************
115 REM * FAKTORIALIS V2 *
120 REM *****************
125 :
130 INPUT "<CLEAR>N=";N
135 S=1
140 FOR I=1 TO N STEP 1
145 S=S*I
150 NEXT I
155 :
160 PRINT "N!=";S
165 END
```

```
100 REM ******************
105 REM * BAS.FAKTORV3    *
110 REM ******************
115 REM * FAKTORIALIS V3 *
120 REM ******************
125 :
130 INPUT "<CLEAR>N=";N
135 S=1
140 FOR I=N TO 1 STEP -1
145 S=S*I
150 NEXT I
155 :
160 PRINT "N!=";S
165 END
```

Eddigi példáink elsősorban a vezérlési strukturák használatát igyekeztek bemutatni. Most következő példáink az eredmények kiírási lehetőségeivel foglalkoznak. A mikrogépek programozásában jártasakat is meglepi a C-16 **PRINT** utasítása, vagy inkább a képernyő szerkesztőnek az a tulajdonsága, hogy idézőjel üzemmódban nem hajtja végre a vezerlő karaktereket, hanem csak akkor, amikor a szóbanforgó sztring kiíródik. Ez teszi lehetővé a képernyő kényelmes szerkesztését. Igy azután a másutt szokásos AT elválasztó jel a **PRINT** utasításban nem használható.

BAS.LABDA
Első példánkban a képernyő két széle között egy ping-pong labda pattog. A hatást úgy érjük el, hogy a kiírandó sztringekbe megfelelő kurzor vezérlő karaktereket helyezünk el. A labda előre mozgatása például a következőképpen történik. Kiírjuk a labdát (a labdának megfelelő karaktert), majd várunk egy kicsit, hogy elég ideig legyen a képernyőn látható. Utána a "<CRSR balra><szóköz>" sztringet írjuk ki, aminek hatására a labda törlődik. Ezt követően a labdát új pozíciójára írjuk ki. Hasonló elven alapul a labda visszafelé mozgatása is.

```
100 REM **************
105 REM * BAS.LABDA    *
110 REM **************
115 REM * TENISZLABDA *
120 REM **************
125 :
130 PRINT"▀▀▀▀▀▀▀▀▀▀▀"
135 :
140 FOR I=1 TO 40
145 PRINT"•";
150 FOR J=1 TO 10: NEXTJ
155 PRINT"‖ ";:NEXT I
160 :
165 FOR I=1 TO 39
170 PRINT"‖ ‖▌•";
175 FOR J=1 TO 10: NEXTJ
180 NEXT I: PRINT"‖";
185 GOTO 140
```

BAS.VELETLEN

A képernyő és színmemóriát használja ez a program, amelyik 1000 CHR$(255) kódu villogó karaktert ír véletlenszerűen a képernyőre. A képernyő pontosan 1000 karaktert tartalmaz, mégsem telik meg. Ez a véletlen kiválasztáson múlik; lehet, hogy egy helyre többször is írunk.

```
100 REM ****************************
105 REM * BAS.VELETLEN              *
110 REM ****************************
115 REM * A PROGRAM VELETLENSZERUEN *
120 REM"* 1000   KARAKTERT RAJZOL   *
125 REM * A KEPERNYORE.             *
130 REM ****************************
135 :
140 PRINT"<CLEAR>";CHR$(14);CHR$(8);
145 FOR I=1 TO 1000
150 K=INT(999*RND(1))
155 :
160 REM A PONT ES SZINENEK BEALLITASA
165 POKE 3072+K,94
170 POKE 2048+K,128
175 NEXT I
180 :
185 GETKEY A$: ?"<CLEAR>"
190 END
```

BAS.MAGAN

A BAS.MAGAN program összeszámolja, hogy egy sztring hány magánhangzót tartalmaz. Ehhez a **MID$** függvényt használjuk. Két egymásba skatulyázott ciklust használunk, az egyikben sorravesszük a sztring karaktereit, a másikban pedig ezt minden egyes magánhangzóval összehasonlitjuk. Ha valamelyikkel megegyezik, akkor egy számlálót megnövelünk eggyel.

```
100 REM ******************************
105 REM * BAS.MAGAN                  *
110 REM ******************************
115 REM * A PROGRAM MEGADJA, HOGY EGY *
120 REM * SZTRING HANY MAGANHANGZOT  *
125 REM * TARTALMAZ                  *
130 REM ******************************
135 :
140 REM M$ TARTALMAZZA AZ OSSZES MAGANHANGZOT
145 M$="AEIOU"
150 :
155 REM A SZTRING BEOLVASASA
160 INPUT "<CLEAR>A$=";A$: S=0
165 :
170 REM A$="" --> NINCS BENNE
175 IF A$="" THEN GOTO 255
180 :
185 REM AZ I CIKLUSBAN AZ A$ MINDEN
190 REM EGYES KARAKTEREROL ELLENORIZZUK,
195 REM HOGY MAGANHANGZO-E.
200 FOR J=1 TO LEN(A$)
205 :
210 REM EGY UJABB (J) CIKLUS ELLENORZI,
215 REM HOGY AZ ADOTT BETU AZONOS-E A
220 REM J.-IK MAGANHANGZOVAL.
225 FOR K=1 TO LEN(M$)
230 IF MID$(A$,J,1)=MID$(M$,K,1) THEN S=S+1: GOTO 240
235 NEXT K
240 NEXT J
245 :
250 REM AZ EREDMENY KIIRASA
255 PRINT"A(Z) ";A$;" SZTRING";S;"MAGANHANGZOT TARTALMAZ."
260 END
```

BAS.FENYUJSAG

A program a fényújságok működését szimulálja: a képernyő közepén ugyanaz a szöveg fut keresztül. A teljes szöveg éppen megjelenítendő részének a kiválasztásához a LEFT$, RIGHT$, illetve a MID$ fuggvényeket használjuk.

```
100 rem ******************************
105 rem * bas.fenyujsag              *
110 rem ******************************
115 rem * az x$-ban tarolt szoveget  *
120 rem * keresztultolja kepernyon,  *
125 rem * mintha kepujsagon latnank. *
130 rem ******************************
135 :
140 rem kezdoertekek beallitasa
145 print chr$(14);chr$(8)
150 s$="                                        ": h=40
155 print"S":x$="A C-16 szemelyi szamitogep igazan kituno!"
160 x$=x$+"    Most vasaroljon!"
165 x$=x$+"    Reklam ar!!!"
170 :
175 rem kiiro ciklus
180 for i=1 to len(x$)+h
185 if i>h goto 200
190 print"Sqqqqqqqqqqqq";mid$(s$,1,h-i)+mid$(x$,1,i)+s$
195 goto 205
200 print "Sqqqqqqqqqqqq";mid$(x$,i-h+1,h)+s$
205 for j=1 to 60: next j
210 next i
215 goto 180
```

BAS.BUVOS

Példánk azt mutatja, hogy a mátrixokat nem csak számolásokra lehet felhasználni. Egy ***N-edrendü* bűvös négyzet** egy olyan NxN-es táblázat, amibe 1-töl N^2-ig úgy írtuk be a számokat, hogy minden oszlopban, sorban és a két átlóban álló elemek összege is egyforma. Páratlan N esetén egyszerü eljárás létezik büvös négyzet előállítására. A számokat 1-től kezdve írjuk be a táblázátba a következá eljárás szerint:

a/ Az 1 az első sor középső oszlopába kerül. (Ehhez kell, hogy N páratlan legyen.)

b/ Ha egy számot már beírtunk valahová, a következő számot általában eggyel kisebb sorba és eggyel nagyobb oszlopba kell beírnunk. Két kivétel van:

1/ Ezzel az eljárással kilépünk a táblából. Ebben az esetben a túloldalon vissza kell lépnünk.

2/ Oda már irtunk egy számot. Ebben az esetben a következő számot ugyanabba az oszlopba, de eggyel nagyobb indexű **sorba** írjuk.

Be lehet bizonyítani, hogy a fenti eljárással mindig bűvös négyzetet kapunk.

A program először beolvassa N értékét, majd a DIM A(N,N) utasítás létrehoz egy (N+1)*(N+1) elemből álló tömböt. Indexelésre az 1..N értékeket használjuk csak. A fentebb vázolt eljárás segitségével kitöltjük a táblázatot, majd kiírjuk a képernyőre. A képernyő mérete miatt nagyobb számok esetén a kiírás már nem szép. A program csak **páratlan** N-re működik helyesen!

```
100 REM ******************
105 REM * BAS.BUVOS      *
110 REM ******************
112 REM * PARATLAN OLDALU *
115 REM * BUVOS NEGYZET   *
120 REM ******************
125 PRINT"<CLEAR>"
130 REM OLDALHOSSZ BEOLVASASA
135 INPUT "N=";N
140 DIM A(N,N)
145 :
150 REM A SZAMOK BEIRASA
155 FOR S=1 TO N^2
160 :
165 IF S=1 THEN I1=1:J1=(N+1)/2: GOTO 210
```

```
170 :
175 I1=I-1: J1=J+1
180 IF I1=0 THEN I1=N
185 IF J1=N+1 THEN J1=1
190 :
195 IF A(I1,J1)=0 THEN GOTO 210
200 I1=I+1: J1=J
205 :
210 A(I1,J1)=S
215 I=I1: J=J1: NEXT S
220 :
225 REM A NEGYZET KIIRASA
230 FOR I=1 TO N
235 FOR J=1 TO N
240 PRINT RIGHT$("    "+STR$(A(I,J)),4);
245 NEXT J: PRINT
250 NEXT I
255 END
```

MAT.RENDEZES

A program az A$(N) tömbben levő sztringeket rendezi sorba. A program négyfajta rendezőalgoritmus szerinti rendezést tartalmaz. Ki lehet próbálni, hogy a gyors rendezés valóban 'gyors'-e? A program a menü kiírásával kezdődik (140-170 sorok), utána választhatjuk ki, melyik eljárással rendezzük sorba a sztringeket (175-180 sorok). Ezután a főprogram meghívja a beolvasó, a kiválasztott rendező, végül pedig a kiíró alprogramot (195-215 sorok). Az egyes algoritmusok közti sebességkülönbség csak N=30 közelében jelentkezik.

```
100 REM ********************
105 REM * MAT.RENDEZES     *
110 REM ********************
115 REM *                  *
120 REM * RENDEZO PROGRAMOK *
125 REM *                  *
130 REM ********************
135 :
140 CLR
145 PRINT"<CLEAR>"
146 PRINT: PRINT: PRINT
148 PRINT"<CTRL RVSON>*** RENDEZO PROGRAMOK ***"
149 PRINT: PRINT
150 PRINT"FELCSERELESES.....1
155 PRINT"BUBOREK...........2
160 PRINT"SHELL-METZNER.....3
165 PRINT"GYORS.............4
```

```
170 PRINT"VEGE..............5
175 INPUT "<CRSR LE>MELYIKET VALASZTJA 1";V
180 IF (V<>INT(V)) OR (V<1) OR (V>5) THEN GOTO 140
185 :
190 IF V=5 THEN PRINT"<CLEAR>";: END
195 GOSUB 530
200 ON V GOSUB 260,315,370,445
205 GOSUB 585
210 GET A$:IF A$="" THEN GOTO 210
215 GOTO 140
220 :
225 REM **************************
230 REM *                        *
235 REM * DIM A$(N) TARTALMAZZA A  *
240 REM *   RENDEZENDO SZTRINGEKET *
245 REM *   A RENDEZES A SZOKASOS  *
250 REM **************************
255 :
260 REM ************************
265 REM * FELCSERELESES RENDEZES*
270 REM ************************
275 FORJ=1TON-1
280 FORK=J+1TON
285 IFA$(J)<=A$(K)THEN GOTO 295
290 TEMP$=A$(J): A$(J)=A$(K):A$(K)=TEMP$
295 NEXTK
300 NEXTJ
305 RETURN
310 :
315 REM ********************
320 REM * BUBOREK-RENDEZES *
325 REM ********************
330 FORJ=N-1TO1 STEP-1:FIN=-1
335 FORK=1TOJ
340 IFA$(K)<=A$(K+1)THEN GOTO 355
345 FIN=0: TE$=A$(K):A$(K)=A$(K+1)
350 A$(K+1)=TE$
355 NEXTK:IF NOT FIN THEN NEXT J
360 RETURN
365 :
370 REM *************************
375 REM * SHELL-METZNER RENDEZES *
380 REM *************************
385 M=N
390 M= INT(M/2)
395 IF M=0 THEN RETURN
400 J=1:K=N-M
405 I=J
410 L=I+M
415 IFA$(I)<=A$(L)THEN GOTO 425
420 TE$=A$(I): A$(I)=A$(L):A$(L)=TE$
```

```
425 I=I-M:IF I>0 THEN 410
430 J=J+1:IF J>K THEN GOTO390
435 GOTO 405
440 :
445 REM ********************
450 REM * GYORS - RENDEZES *
455 REM ********************
460 DIM ST((LOG(N)/LOG(2)+4),1)
465 S=1:ST(1,0)=1:ST(1,1)=N
470 L=ST(S,0):R=ST(S,1):S=S-1
475 J=L:K=R: X$=A$((L+R)/2)
480 IF A$(J)<X$ THEN J=J+1: GOTO 480
485 FOR V=1 TO 1E6: IF A$(K)>X$ THEN K=K-1: NEXT
490 IF J=K THEN J=J+1: K=K-1: GOTO 480
495 IF J<K THEN:  TEMP$=A$(J):  A$(J)=A$(K):A$(K)=TEMP$:  J=J+1:
K=K-1: GOTO 480
500 IFJ<R THEN S=S+1: ST(S,0)=J: ST(S,1)=R
505 R=K
510 IF L<R THEN 475
515 IF S>0 THEN 470
520 RETURN
525 :
530 REM ******************
535 REM * BEOLVASO RUTIN *
540 REM ******************
545 PRINT"<CLEAR>"
550 INPUT "A RENDEZENDO ELEMEK SZAMA (N>2)";N
555 DIM A$(N)
560 FOR I=1 TO N
565 INPUT "KOVETKEZO SZTRING=";A$(I)
570 NEXT I
575 RETURN
580 :
585 REM ***************
590 REM * KIIRO RUTIN *
595 REM ***************
600 FOR I=1 TO N
605 PRINTA$(I)
610 NEXT I
615 PRINT"<RVSON>NYOMJ MEG EGY BILLENTYUT !"
620 RETURN
```

7.2 Grafikus utasítások használata

Ebben a paragrafusban a C-16 grafikus utasításainak használatára mutatunk példákat. Az ebbe a csoportba tartozó BASIC utasításoknál is adtunk már teljes példákat, így ez a paragrafus mintegy kiegésziti a 3.3 fejezetben levő - igaz kissebb példákat. Az itt található rajzoló programok mindegyike úgy lett megtervezve, hogy a kész rajzot egészen addig látjuk, míg egy billentyűt meg nem nyomunk. Azután újból a karakteres képernyő kerül kijelzésre.

SZINEK
A program a képernyőn bemutatja a C-16 által generálható valamennyi színt. A külső 10-60 ciklus változója (J) állítja a színeket, míg a belső 10-40 ciklus változója (I) állítja a szín intenzitását. Fekete-fehér képernyőn természetesen csak a szürke különféle árnyalatai látszanak.

```
3 A$="R                                        "
5 FOR J=1 TO 16
6 COLOR 4,1: COLOR 0,1
10 FOR I=0 TO 7
20 COLOR 1,J,I
30 PRINTA$;A$;A$
40 NEXT I
50 FOR I=1 TO 2000: NEXT I
60 PRINT"<CLEAR>";: NEXT J
```

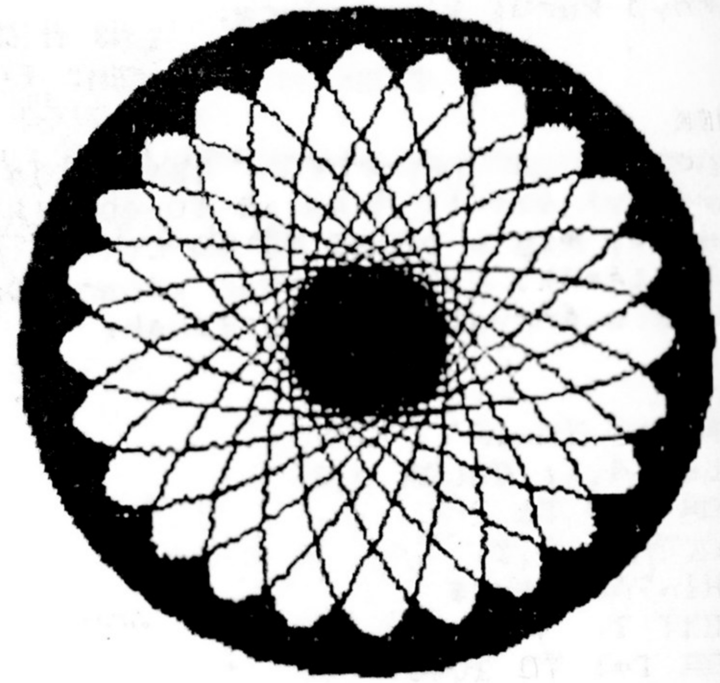
GYURU
A program egy körgyűrűt rajzol, majd a PAINT utasítás felhasználásával azt besatírozza. A satírozás közben jól megfigyelhető az a logika, ahogy a PAINT utasítás végrehajtása történik. (Lásd a 137. oldal ábráját!)

```
10 GRAPHIC 1,1
15 CIRCLE 1,159,100,40,40
20 CIRCLE 1,159,100,80,80
30 PAINT 1,159,50
40 PAINT 1,159,100
50 PAINT 0,159,100
60 GETKEY A$
70 GRAPHIC 0
```

VIRAG
Az alábbi példa mutatja, hogy a legegyszerűbb utasítások felhasználásával is igen szép rajzok készithetők. Példánk a CIRCLE utasítás azon két tulajdonságán alapul, hogy egyrészt

lehetőség nyílik ellipszis rajzolására, másrészt az ellipszis tengelyének hajlásszögét magunk adhatjuk meg. Egy további CIRCLE, illetve két PAINT utasítás végrehajtása után kész is van a *virág*!

```
10 GRAPHIC 1,1
20 FOR I=0 TO 11
30 CIRCLE 1,159,100,20,80,,,15*I
40 NEXT I
50 PAINT 1,159,100
60 CIRCLE 1,159,100,90,90
70 PAINT 1,70,100
80 GETKEY A$
90 GRAPHIC 0
```



SZINUSZ

Az alábbi példaprogram megfelelője szinte valamennyi nagyfelbontású grafikát kezelő gép gépkönyvében szerepel. Feladata egyszerű: egy színuszgörbét rajzol a képernyőre. A rajzolást az 50-80 ciklus végzi, amelyikben a DRAW utasítás segítségével a színuszgörbe utoljára megrajzolt pontját összekötjük a görbe következő pontjával. Ilyen módon a görbe a *meredekebb* szakaszain is folytonos lesz. (Érdemes kicserélni a 70. sort a DRAW 1,X,Y utasításra és megfigyyelni a különbséget!)

```
10 COLOR 0,1
20 COLOR 1,2
30 GRAPHIC 1,1
40 LOCATE 0,100
50 FOR X=1 TO 319
60 Y=INT(100+99*SIN(X/20))
70 DRAW 1 TO X,Y
80 NEXT X
90 FOR L=1 TO 5000
100 NEXT L
110 GRAPHIC 0
```

LEGGOMB

Programunk a sprite-ok használatát mutatja be. A DATA sorokban helyeztük el egy léggömb adatait, amit a 30-90 ciklusban beolvasunk, s a 80. sorban a nagyfelbontású képernyő megfelelő helyére irjuk be. A ciklus végére megjelenik a képernyőn a teljes léggömb. Következő lépésként a léggömb alakját a 100 sorban levő SSHAPE utasitás segitségével az X$ változóba mentjük el.

Ezzel azonban csak annyit tettünk, hogy a léggömb alakját a C-16 számára kezelhető alakban állítottuk elő. A továbbiakban a GSHAPE utasítás segítségével a képernyő átlója mellett elkezdjük a léggömb mozgatását. Ezt a 110-140 ciklus végzi. A ciklusban két GSHAPE utasítás szerepel. Egyik a léggömb1 utolsó megjelenésének törlését végzi, a másik a képernyőre rajzolja a léggömb új helyét. A törléshez a (120. sorban) a GSHAPE 4. paraméterét is használnunk kell. A 4-es érték a *kizáró vagynak* felel meg.

```
0 GRAPHIC 1,1
20 G=2*4096
30 FOR I=0 TO 62
40 READ A
50 S=INT(I/3):O=I-3*S
60 SS=INT(S/8):SO=S-8*SS
70 CI=G+320*SS+SO+8*O
80 POKE CI,A
90 NEXT I
95 :
100 SSHAPE X$,0,0,24,21
110 FOR X=10 TO 320 STEP 10
120 GSHAPE X$,X-10,X/2-5,4
130 GSHAPE X$,X,X/2
140 NEXT X
150 GETKEY A$: GRAPHICO:END
155 :
190 DATA 0,127,0,1,255,192,3,255,224,3,227,224
200 DATA 7,217,240,7,223,240,7,217,240,3,231,224
210 DATA 3,255,224,3,255,224,2,255,160,1,127,64
220 DATA 1,62,64,0,156,128,0,156,128,0,73,0,0,73,0
230 DATA 0,62,0,0,62,0,0,62,0,0,28,0
```

SOKLEGGOMB

A program az előző program módositása. Nem mozgatjuk a léggömböt, hanem az X$-ban levő képelem segitségével - véletlenszerűen - telerajzoljuk a képernyőt különböző színű léggömbökkel.

```
10 COLOR 0,14,5 : GRAPHIC 1,1
20 COLOR 4,14,5
30 G=2*4096
40 FOR I=0 TO 62
50 READ A
60 S=INT(I/3):O=I-3*S
70 SS=INT(S/8):SO=S-8*SS
80 CI=G+320*SS+SO+8*O
90 POKE CI,A
100 NEXT I
110 :
120 SZ(1)=2:SZ(2)=3:SZ(3)=10 :SZ(4)=6:SZ(5)=1
130 SSHAPE X$,0,0,24,21
140 FOR I=0 TO 70
150 X=RND(0)*310: Y=RND(0)*190
160 J=INT(RND(0)*3.999999+1)
170 IN=INT(RND(0)*6.999999+1)
180 COLOR 1,J,IN
190 GSHAPE X$,X,Y,2
200 NEXT I
210 GETKEY A$: GRAPHIC0:END
220 :
230 DATA0,127,0,1,255,192,3,255,224,3,227,224
240 DATA 7,217,240,7,223,240,7,217,240,3,231,224
250 DATA 3,255,224,3,255,224,2,255,160,1,127,64
260 DATA 1,62,64,0,156,128,0,156,128,0,73,0,0,73,0
270 DATA 0,62,0,0,62,0,0,62,0,0,28,0
```

RAJZOLO

A képernyőn olyan lehetőségeink vannak, amit papíron még sokáig nem tudunk visszaadni: pl. a 121 féle szín. Mégis jó dolog ha a képernyőn látható rajzról egy papírmásolatot (egy ún. hard-copy-t) tudunk készíteni. Az alábbi program ezt a feladatot oldja meg: a nagyfelbontású képernyő tartalmát kiírja egy MPS801, vagy vele funkcionálisan ekvivalens nyomtatóra. (A grafikus nyomtatók nem kompatibilisek egymással, ezért elő-fordulhat, hogy egyes nyomtatókkal a program nem használható.)

```
100 REM ***************************
105 REM * NAGYFELBONTASU KEPERNYO *
110 REM *       KINYOMTATASA      *
115 REM ***************************
120 OPEN 4,4: DIM Z%(199)
135 S=8192
140 FOR I=39 TO 0 STEP -1:FOR J=0 TO 24
145 FOR K=0 TO 7
150 Q=J*8+K
155 Z%(Q)=Z%(Q)+(PEEK(S+320*J+I*8+K))*2^Y
160 A$=A$+CHR$((Z%(Q) AND 127) + 128)
```

```
165 Z%(Q)=Z%(Q)/128
170 NEXT K:NEXT J
175 PRINT#4,CHR$(8);A$:A$=""
180 Y=Y+1:IF Y=7 THEN Y=0:GOTO 195
185 PRINTI"   q":NEXT I
190 R=1
195 FOR L=0 TO 199
200 A$=A$+CHR$((Z%(L) AND 127)+ 128)
205 Z%(L)=Z%(L)/128:NEXT L
210 PRINT#4,CHR$(A);A$:A$=""
215 IF R=0 THEN 185
220 CLOSE 4
225 END
```

AUTO

A program a botkormány és a grafika együttes használatára mutat példát. A képernyő jobb elsó sarkában levő gépkocsit a botkormány segítségével kormányozhatjuk. Ha benyomjuk a <TŰZ> billentyűt, a kocsi gyorsabban halad. A játékprogramok hasonlóan mozgatják a különféle tárgyakat, állatokat és embereket, de mivel gépi kódban vannak megírva, ezért sokkal gyorsabbak.

```
100 REM ADATOK BEOLVASASA
110 FOR I=1 TO 60: READ X: A$=A$+CHR$(X): NEXT I
120 FOR I=1 TO 8: READ DX(I):NEXT I
130 FOR I=1 TO 8: READ DY(I):NEXT I
140 :
150 COLOR 0,14,5
160 COLOR 4,14,5
170 GRAPHIC1,1
180 :
190 X=290:Y=170: GSHAPE A$,X,Y,2
200 :
210 DO
220 M=JOY(1): IR=M AND 127: S=(M AND 128)/128+1:  IF  IR=0  THEN
GOTO 280
230 XU=X+DX(IR)*S: XU=XU-INT(XU/319)*319
240 YU=Y+DY(IR)*S: YU=YU-INT(YU/199)*199
250 GSHAPE A$,X,Y,4
260 GSHAPE A$,XU,YU,4
270 X=XU:Y=YU
280 GETKEY U$
290 LOOP UNTIL U$=CHR$(13)
300 :
310 GRAPHIC 0: END
320 :
```

```
330 REM AZ AUTO ADATAI
340 DATA 0,0,0,0,0,8,0,0,0,31,248,0,0,99,12,0,1,131,12,0,7,3,
        6,0,12,3,6,0
350 DATA 255,255,255,0,255,243,255,0,255,255,255,0,109,255,
        182,0,30
360 DATA 0,120,0,30,0,120,0,12,0,48,0,24,0,13,0
370 REM AZ IRANYOK ADATAI
380 :
390 DATA 0,3.5,5,3.5,0,-3.5,-5,-3.5
400 DATA -5,-3.5,0,3.5,5,3.5,0,-3.5
```

Mint látható a C-16 grafikus utasításaival nagyon sok ábrát, rajzot tudunk elkészíteni. Vannak olyan rajzok is azonban amit ezekkel az eszközökkel nem is olyan egyszerű elkészíteni. Egyik ilyen pl. a fenti példákban szereplő léggömb alakja. Hogyan állíthatjuk elő a LEGGOMB programban szereplő DATA utasításokat?. Másik példa mondjuk egy egyenlő oldalú háromszög. Ahhoz, hogy azt pl. a DRAW utasítás segítségével megrajzoljuk ki kellene számítani a csúcspontok koordinátáit.

A fejezet hátralevő részében a rajzolást tovább könnyitő segédprogramokat, *utility-ket* mutatunk be. Ilyennek tekinthető a fentebb ismertetett RAJZOLO program is.

TEKNOSBEKA GRAFIKA

A teknősbéka grafika lényege, hogy nem abszolút, hanem relativ utasításokkal dolgozik. A különbség a következő. Ha a C-16 meglévő utasításaival akarunk egy szakaszt megrajzolni, akkor a szakasz **két végpontjának** koordinátáit, azok valódi értékét kell megadnunk. Ugyanezt a szakaszt azonban úgy is megrajzolhatjuk, hogy az egyik végpontjából megfelelő **irányú** és megfelelő **hosszú** szakaszt rajzolunk. (A matematikát kedvelők számára azt is mondhatjuk, hogy ebben az esetben a szakaszt vektornak tekintjük, s a szakasz megrajzolása az adott vektor, adott pontból való felmérését jelenti.)

Egy kicsit részletesebben a teknősbéka geometri használata a következőt jelenti. Képzeljünk el egy teknösbékát, amelyik a szájában egy ceruzával a képernyőn mászik. Ha a teknősbéka felemelt fejjel közlekedik, akkor nem látszik meg a nyoma, de ha lógó fejjel mászik, akkor amerre jár egy vékony vonalat húz a képernyőn. A teknősbéka grafikai utasításai a teknősbéka mozgását szabályozzák. **Valamennyi utasítás relativ, tehát a teknősbéka utolsó helyéhez viszonyított elmozdulást kell majd megadnunk.** Lehetőség van az elmozdulás irányának, illetve hosszának megadására.

A C-16 esetében egyszerű BASIC alprogramokkal valósíthatjuk meg a teknősbéka grafikát. Az alábbi program - meglehetősen bőbeszédűen - sorolja fel ezeket a rutinokat. Ezért megadjuk ezeknek egy tömörített változatát is, ami l4nyegesen több szabad helyet hagy a memóriában, s azt a változatot használjuk majd a teknősbéka grafika használatának bemutatására.

Vegyük sorra a teknösbéka grafika alprogramjait! (Mindenütt sorra feltüntettük a LOGO nyelv megfelelő utasításainak a nevét is.)

1/ A képernyő törlése. értelemszerű
2/ A ceruza letevése. A PD változót használjuk annak jelzésére, hogy a teknős felemelt fejjel mászik-e vagy sem. Ha igen, akkor PD=0, ha nem, akkor PD=1. **A teknős útja csak akkor látszik, ha a fejét letesszük.**
3/ A ceruza felemelése. Lásd 2/.
4/ A teknősbékának a képernyő közepére mozgatása. A teknősbéka aktuális helyzetét az XT,YT koordinátapár tartalmazza. Az alprogram ennek értékét a képernyő közepére állítja, majd végrehajtja a megfelelő mozgást. Az alprogram egyben a teknősbéka fejét is alapállásba hozza.
5/ Az ALFA nevű változó tartalmazza a teknösbéka fejének irányát szögekben mérve. A teknős fejének jobbra, illetve balra forgatása az ALFA változó értékének változtatását jelenti.
6/ Lehetőség van a teknős előre, illetve hátra mozgatására (FORWARD, BACKWARD). Ehhez első lépésben a teknős új helyzetének koordinátáit kell kiszámítani, figyelembe véve, hogy az ALFA szögben van megadva. Következő lépésként a teknőst erre a pontra kell mozgatnunk. Ha lennt van a feje, akkor a DRAW, ha nincs akkor a LOCATE utasítást használjuk.
7/ Végül a DRAW (rajzol!) alprogram inicializálja a teknősbéka grafikát: törli a grafikus képernyőt, középre állítja a teknőst, illetve leteszi a teknős fejét.

```
10 GOTO 2000
1000 REM ***********************
1010 REM * TEKNOSBEKA GRAFIKA *
1020 REM ***********************
1030 :
1040 :
1050 REM #######################
1060 REM # CLEARSCREEN         #
1070 REM #                     #
1080 REM # A KEPERNYO TORLESE #
1090 REM #######################
1100 :
1110 GRAPHIC 1,1
1120 RETURN
1130 :
```

```
1140 REM ###################
1150 REM # PENDAWN         #
1160 REM #                 #
1170 REM # CERUZA LETEVESE #
1180 REM ###################
1190 :
1200 PD=1: RETURN
1210 :
1220 REM #######################
1230 REM # PENUP               #
1240 REM #                     #
1250 REM # CERUZA FELEMELESE   #
1260 REM #######################
1270 :
1280 PD=0: RETURN
1290 :
1300 REM ##############################
1310 REM # HOME                       #
1320 REM #                            #
1330 REM # KEPERNYO KOZEPERE MOZGAS #
1340 REM ##############################
1350 :
1360 ALFA=-90
1360 XT=159:YT=100: GOTO 1615
1380 :
1390 REM ##########
1400 REM # RIGHT  #
1410 REM #        #
1420 REM # JOBBRA #
1430 REM ##########
1440 :
1450 ALFA=INT((ALFA-SZOG)/360)
1460 RETURN
1470 REM ########
1480 REM # LEFT  #
1490 REM #       #
1500 REM # BALRA #
1510 REM ########
1520 :
1530 ALFA=INT((ALFA+SZOG)/360)
1540 RETURN
1550 :
1560 REM ##########
1570 REM # FORWARD #
1580 REM #         #
1590 REM # ELORE   #
1600 REM ##########
1610 DEL=<pi>/180*ALFA
1611 XT=XT+HOSSZ*COS(DEL): YT=YT+HOSSZ*SIN(DEL)
1615 IF PD=0 THEN GOTO 1635
1620 DRAW 1 TO XT,YT
```

```
1630 RETURN
1635 LOCATE XT,YT
1636 RETURN
1640 :
1646 :
1650 REM ###########
1660 REM # BACKWARD #
1670 REM #          #
1680 REM # HATRA    #
1690 REM ###########
1700 DEL=<pi>/180*ALFA
1705 XT=XT-HOSSZ*COS(DEL): YT=YT-HOSSZ*SIN(DEL)
1708 IF PD=0 THEN GOTO 1725
1710 DRAW 1 TO XT,YT
1720 RETURN
1725 LOCATE XT,YT
1728 RETURN
1730 :
1740 REM ###########
1750 REM # DRAW     #
1760 REM #          #
1770 REM # RAJZOLAS #
1780 REM ###########
1790 :
1800 GOSUB 1110: LOCATE 159,100
1810 ALFA=-90: PD=1: XT=159: YT=100
1820 RETURN
2000 REM ITT KEZDODIK A FOPROGRAM
```

TEKNOS.RUTINOK

Az alábbi program a fenti tömörített változata. Külön táblázatban foglaltuk össze az egyes alprogramok feladatát, illetve a meghívásukhoz szükséges GOSUB utasításban szerepeltetendő sorszámot.

Funkció	**Belépési pont**	**Parameter**
CLEARSCREEN	20	
PENDAWN	30	
PENUP	40	
HOME	50	
RIGHT	60	SZOG
LEFT	70	SZOG
FORWARD	80	HOSSZ
BACKWARD	130	HOSSZ
DRAW	170	

```
10 GOTO 190
20 GRAPHIC 1,1:RETURN
30 PD=1: RETURN
40 PD=0: RETURN
50 DRAW 1 TO 159,100:XT=159:YT=100
60 ALFA=-90:RETURN
70 ALFA=ALFA-SZOG: RETURN
80 ALFA=ALFA+SZOG: RETURN
90 DEL=<pi>/180*ALFA
100 XT=XT+HOSSZ*COS(DEL): YT=YT+HOSSZ*SIN(DEL)
110 IF PD=1 THEN DRAW 1 TO XT,YT :ELSE LOCATE XT,YT
120 RETURN
130 DEL=<pi>/180*ALFA
140 XT=XT-HOSSZ*COS(DEL): YT=YT-HOSSZ*SIN(DEL)
150 IF PD=0 THEN LOCATE XT,YT: ELSE DRAW 1 TO XT,YT
160 RETURN
170 GOSUB 20: LOCATE 159,100
180 ALFA=-90: PD=1: XT=159: YT=100: RETURN
190 REM FOPROGRAM
```

A teknősbéka grafikát használó programok írása előtt a fenti programot mindig be kell töltenünk a memóriába, s a főprogramot a 200. sórtól kezdve kell elkezdenünk megírni. A továbbiakban három példát mutatunk a teknősbéka grafika alprogramjainak használatára.

SOKSZOG

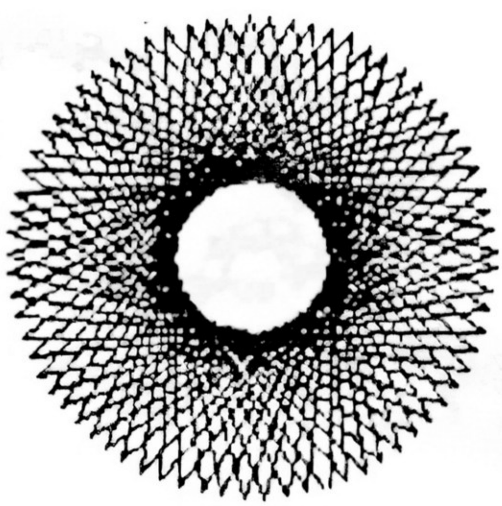
A program segítségével tetszőleges oldalszámú és oldalhosszú szabályos sokszöget tudunk rajzolni. A megoldást legegyszerűbben a háromszög példáján képzelhetjük el. Például egy 70 oldalhosszú háromszöget ugy rajzolhatunk, hogy előreléptejük a teknőst 70-nel, majd elforditjuk 120 fokkal, és ezt mmég kétszer megismételjük. Az alábbbi program ennek az általánositása.

A 240. sorban beolvassuk az oldalszámot és az oldal hosszát. A 250. sor inicializálja a teknősbéka grafikát. Egyben itt számitjuk ki, hogy egy-egy forduláskor mennyit is kell fordulni. A 260-280 ciklusban N-szer léptetjük előre és forditjuk el a teknőst. (Megjegyezzük, hogy a C-16 esetében a CIRCLE utasítással is rajzolhatunk szabályos sokszöget. Sőt akár affin szabályos sokszöget is!)

```
200 REM ###########
210 REM # SOKSZOG #
220 REM ###########
230 :
240 INPUT "<CLEAR><CRSR LE>N,HOSSZ=";N,HOSSZ
250 GOSUB 170: SZOG=360/N
260 FOR I=1 TO N
270 GOSUB 80: GOSUB 60
280 NEXT I
290 GETKEY A$: GRAPHIC 0
```

Az alábbi csillagrajzoló program a fenti SOKSZOG *elrontásán* alapul. A szabályos sokszög rajzolásának épp az a lényege, hogy a SZOG pontosan 360/N. Mi történik vajon akkor, ha mégsem. A következő programban a SZOG értékét mi magunk adhatjuk meg. Próbáljuk ki az N=73: SZOG=143: HOSSZ=90 értékeket!

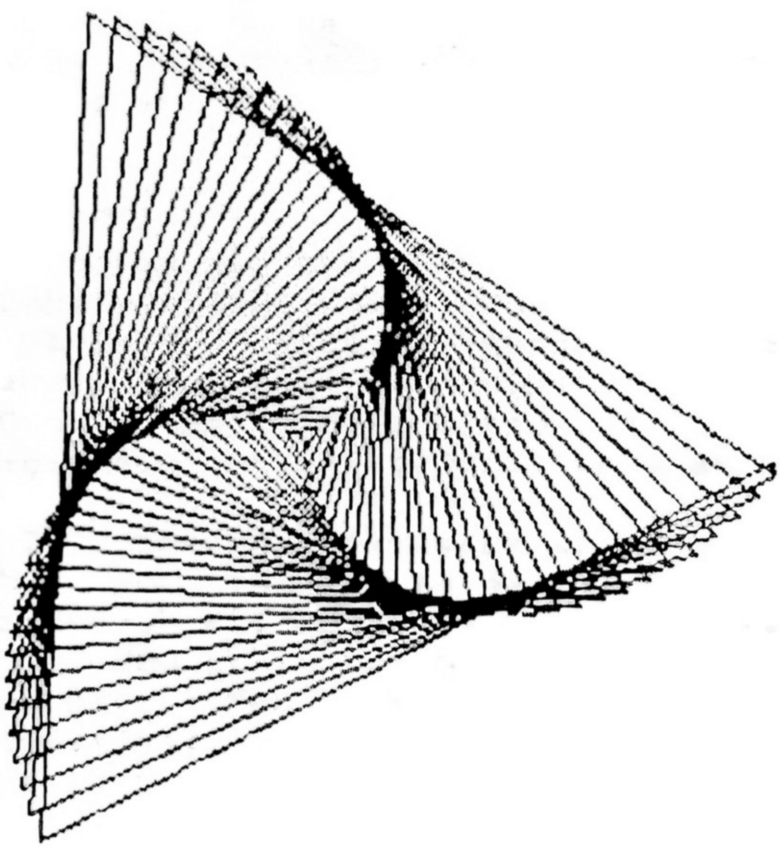
```
200 REM ###########
210 REM # CSILLAG #
220 REM ###########
230 :
240 INPUT "<CLEAR><CRSR LE>N,HOSSZ,SZOG=";N,HOSSZ,SZOG
250 GOSUB 170
260 FOR I=1 TO N
270 GOSUB 80: GOSUB 60
280 NEXT I
290 GETKEY A$: GRAPHIC 0
```



```
200 REM ##########
210 REM # SPIRAL #
220 REM ##########
230 :
240 INPUT "<CLEAR><CRSR LE>NOV,SZOG=";NO,SZ
250 GOSUB 170: HO=1
260 DO: GOSUB 60
270 HO=HO+NOV: GOSUB 80
280 LOOP UNTIL HO>220
290 GETKEY A$: GRAPHIC 0
```

SPIRAL

A spirálrajzoló program a CSILLAG módosításán alapul. A csillag program alapját az képezte, hogy az elfordulás nagysága mindig ugyanannyi. Vajon mi történik akkor, ha ezt az értéket folyamatosan ugyanazzal a számmal növeljük? Az alábbi program épen így működik. Próbaként használjuk a NOV=2: SZOG=121 értékeket.



Sprite editor

Az alábbi BASIC program lehetővé teszi 24*21 nagyságú sprite-ok szerkesztését, illetve a programokban elhelyezendő DATA konstansok kiíratását. Ezzel a programmal kiszámított értékeket használtuk pl. a LEGGOMB program DATA utasításaiban.

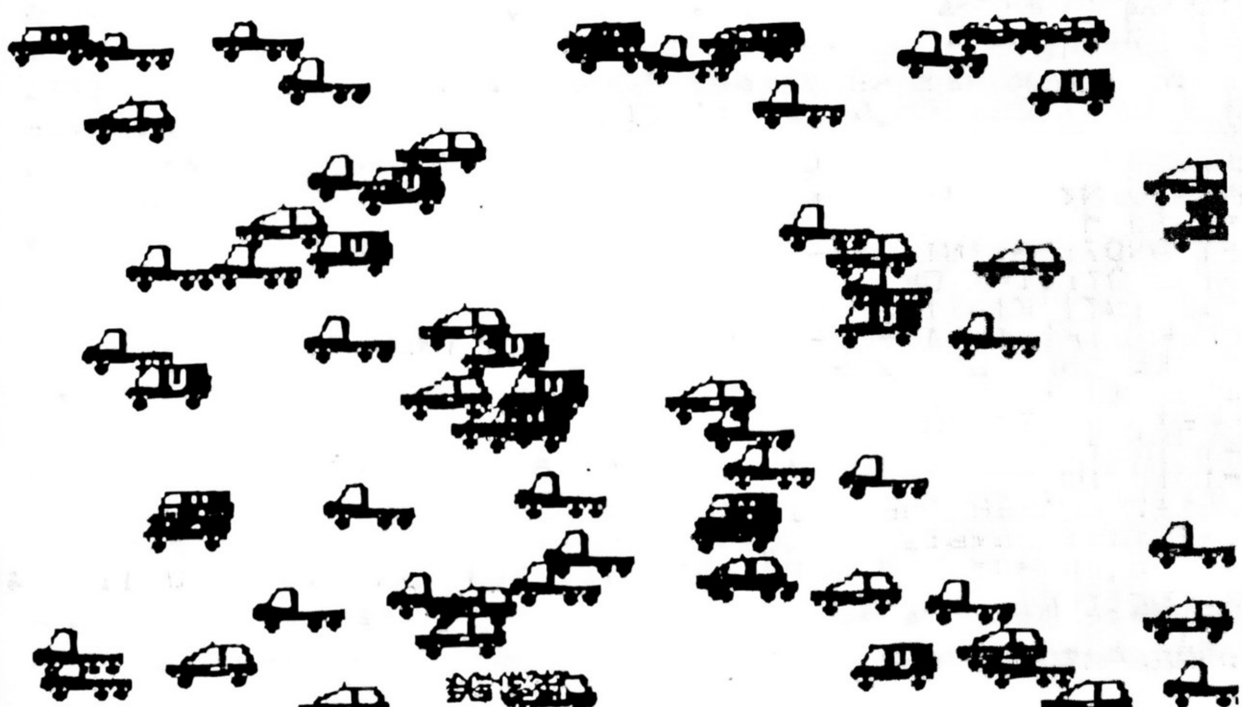
A program betöltés után a képernyő bal felső sarkába berajzolja azt a területet, ahová a képet megszerkeszthetjük. Ezen a területen a kurzor, a <HOME> és a <RETURN> billentyűk segítségével mozoghatunk. A bekapcsolt pontoknak egy-egy '*' karakter felel meg. A pontok be, illetve kikapcsolására a <.>, illetve a <szóköz> billentyűket használhatjuk. A megszerkesztett kép inverzét a <RVS ON> billentyű lenyomásával kapjuk meg.

A kép megszerkesztése után a V billentyű lenyomásával a nagyfelbontású képernyőn is megnézhetjük a sprite-ot. Ekkor írja ki a program az SSHAPE utasítással kapott sztring karaktereinek ASCII kódját is. Bármely billentyű megnyomásával visszatérhetünk az alakzat szerkesztéséhez.

A kapott adatokat egy tetszőleges programban a DATA utasításokban kell elhelyezni. Ezután a következő programrész segítségével előállíthatjuk a már egyszer megszerkesztett képet:

```
1000 KE$="": FOR I=1 TO 92
1010 READ A: KE$=KE$+CHR$(A): NEXT I
```

Az alábbi ábrán látható autók alakját ezzel a programmal szerkesztettuk meg:



A sprite editor program listája:

```
100 L$="QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ"
110 K$="]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]"
120 KK=3072: NK=8192
130 PRINT"Z";CHR$(8);
140 N=21:M=24
150 FOR I=1 TO N
160 FOR J=1 TO M:PRINT" ";:NEXT J
170 PRINT"|"
180 NEXT I
190 FOR J=1 TO M:PRINT"_";: NEXT J: PRINT"_S";
200 I=1:J=1:PRINT"R?_|";
210 GET A$: IF A$="" THEN GOTO210
220 IF A$="." THEN GOTO 460
230 IF A$=" " THEN GOTO 480
240 HE=KK+(I-1)*40+(J-1)
250 J$=" ": IF (PEEK(HE)AND127)=42 THEN J$="*"
260 IF A$="O" THEN I=I-1: GOTO 340
270 IF A$="Q" THEN I=I+1: GOTO 340
280 IF A$="]" THEN J=J+1: GOTO 340
290 IF A$="|" THEN J=J-1: GOTO 340
300 IF A$="V" THEN GOTO 560
310 IF A$=CHR$(13) THEN I=I+1:J=1:GOTO 340
320 IF A$="N" THEN GOTO 500
330 IF A$="S" THEN I=1:J=1: GOTO 340
340 IF I>N THEN I=1
350 IF I=0 THEN I=N
360 IF J>M THEN J=1
370 IF J=0 THEN J=M
380 PRINTJ$;
390 PRINT"S";:IF I=1 THEN GOTO 410
400 PRINT MID$(L$,1,I-1);
410 IF J=1 THEN GOTO 430
420 PRINT MID$(K$,1,J-1);
430 HE=KK+(I-1)*40+(J-1)
440 J$="R?_|": IF (PEEK(HE)AND127)=42 THEN J$="R*_|"
450 PRINTJ$;:GOTO 210
460 J$="*":J=J+1:IF J>M THEN J=1
470 GOTO 340
480 J$=" ":J=J+1:IF J>M THEN J=1
490 GOTO 340
500 PRINT"S";: FOR II=1 TO N
510 FOR JJ=1 TO M
520 HE=KK+(II-1)*40+(JJ-1)
530 J$="*": IF (PEEK(HE)AND127)=42 THEN J$=" "
540 PRINTJ$;: NEXT JJ: PRINT: NEXT II
550 GOTO 390
560 COLOR 1,1
570 FOR KI=1 TO N
580 FOR KJ=1 TO M
590 SR=(KI-1)AND7:OS=INT((KJ-1)/8)
600 OK=(KJ-1)AND7:BI=7-OK
610 HE=NK+320*INT((KI-1)/8)+8*OS+SR
620 IF (PEEK(KK+(KI-1)*40+KJ-1)AND 127)<>42 THEN GOTO 640
630 POKE HE,PEEK(HE) OR (2^BI)
640 NEXT KJ: NEXT KI
650 SSHAPE KE$,0,0,23,20
660 FOR KI=1 TO 7
670 FOR KJ=1 TO 10
680 IF (KI-1)*10+KJ=68 THEN KJ=10:GOTO 700
690 CHAR 1,4*(KJ-1),6+KI,
             RIGHT$("    "+STR$(ASC(MID$(KE$,(KI-1)*10+KJ,1))),4)
700 NEXT KJ: NEXT KI
710 POKE 239,0
720 GET A$: GRAPHIC 0: GOTO 210
```

8. fejezet

A gépi kódú monitor használata

A C-16 számítógép egy TEDMON nevű monitorral rendelkezik. A BASIC-ből a monitort a

MONITOR

parancs kiadásával érhetjük el. Ezt követően – egészen a monitorból való kilépésig – nem használhatjuk a BASIC utasításokat.

A gépi kódú monitor a processzor regisztereinek, illetve a memóriacímek közvetlen írását, olvasását, a memória adott részének elmentését, visszatöltését; továbbá egyszerű assembly nyelvű programozást tesz lehetővé. Ebben a fejezetben a gépi kódú monitor használatát ismertetjük. Nem célunk a processzor utasításkészletének ismertetése, hiszen azt már mások sok helyen megtették. A 7501-es processzor szoftver kompatibilis az M6510-es processzorral. A részletek iránt érdeklődők ezért a szükséges ismereteket megszerezhetik **dr. Úry László: *Commodore 64 BASIC felhasználói kézikönyv*, vagy Erdős Iván: *Commodore 64 assembly programozás* című, az LSI ATSZ gondozásában megjelent könyvekből.**

A TEDMON parancsai egyetlen karakterből állnak, amit – szóközzel elválasztva – követnek a paraméterek. A parancsok kiadására a C-16 teljes képernyős szerkesztője használható. A paraméterek sztringkonstansok vagy hexadecimális számok. Ez utóbbiak a paraméter jellegétől függően vagy **pontosan 2**, vagy **pontosan 4** hexadecimális számból állnak. A számok elé a $ jelet nem szabad kiírni!

A

Helyben fordítás. Az 'A' a '.'-tal helyettesíthető

Szintaxis: A <cím> <mnemonik> [<operandus>]

A parancs kiadása után a monitor a parancsban szereplő assembly nyelvű utasítást lefordítja, s a neki megfelelő gépi kódot a

<cím>-től kezdődően elhelyezi. A következő sorban megjelenik egy

A <újcím>

alakú üzenet, ahol <újcím> a fordítás és az elhelyezés utáni első szabad memória címe. Az assembly utasitások így folyamatosan írhatók.

A <mnemonik> az M6510-es processzor szokásos mnemonikjainak bármelyike lehet. Az <operandus> azonban nem tartalmazhat semmilyen kifejezést vagy címkét. Egyetlen könnyebbség, hogy relatív ugrások esetén is az **abszolút címet kell beírni.** A monitor azután kiszámítja a különbséget.

Példák: A 1400 LDX #$03
A 1402

C

Memóriaterületek összehasonlítása.

Szintaxis: C <cím1> <cím2> <cím3>

A parancs a <cím1>-től <cím2>-ig tartó memóriarész tartalmát hasonlítja össze a <cím3>-tól kezdődő hasonló hosszú memóriarésszel. A <cím2> memóriarész tartalma **már** nem szerepel az összehasonlításban.

Példa: C 2000 2501 4000

D

A memória adott helyén levő gépi kódú utasítást mnemonik alakban állítja elő.

Szintaxis: D [<cím1>] [<cím2>]
A parancs a megadott memóriacímek közti tartományban levő gépi kódú programot fordítja vissza mnemonikos alakúvá. Ha értelmezhetetlen kódot talál, akkor *???* választ ad:

Példák: D 3000 3004

```
. 3000  A9 00   LDA #$00
. 3002  FF      ???
. 3003  D0 2B   BNE $3030
```

A visszafordított sorok előtti . lehetővé teszi, hogy azonnal módosíthassuk és újrafordíthassuk az utasítást.

F

Karakterfeltöltés.

Szintaxis: F <cím1> <cím2> <érték>

A monitor a memória <cím1>-től <cím2>-ig terjedő részét az <érték> byte-tal tölti fel. A <cím2>-be magába már nem töltődik be ez az érték. Az érték két byte-os hexadecimális szám lehet csak.

Példák: F 0400 0518 EA

G

Elindít egy gépi kódú programot.

Szintaxis: G [<cím>]

A vezérlés a <cím>-nél kezdődő gépi kódú rutinra kerül. Egészen egy gépi kódú RTS vagy BRK végrehajtásáig a vezérlés kikerül a gépi kódú monitor felügyelete alól.

H

Adott karaktersorozat keresése.

Szintaxis: H <cím1> <cím2> <adatok>

A monitor a <cím1>-től a <cím2>-ig terjedő memóriarészben keres az <adatok>-nak megfelelő byte sorozatot. Ha ilyent talál, a sorozat első elemének címét kiírja a képernyőre, majd folytatja a keresést. Az adatok vagy egymástól szóközzel elválasztott hexadecimális számok, vagy sztringkonstansok lehetnek.

Példák: H 8000 9000 "KUKAC"
H 2000 2400 EE EE EE FA

L

PRG típusú file betöltése.

Szintaxis: L <filenév> <egységszám>

A monitor a file első két byte-ja által meghatározott címtől

kezdve betölti a programot.

Példák: L "ALAKZAT" 08 (töltés lemezről)
L "ADATOK" 01 (töltés kazettáról)

M

Kiírja a memória tartalmát.

Szintaxis: M [<cím1>] [<cím2>]

A parancs a <cím1>-től kezdődően kiírja a memória tartalmát mind hexadecimális, mind karakteres alakban. Ha a második paraméter elmarad, akkor a következő 96 memóriarekesz tartalmát írja ki. A kiírás formátuma megegyezik a > parancs formájával, így lehetőség nyílik a memória tartalmának azonnali módosítására.

Példák: M A048 A048

*>A048 41 E7 00 AA AA 00 98 56 :A!.**..V*

R

Kiírja a regiszterek tartalmát.

Szintaxis: R

A parancs kiírja az utasításszámláló, az állapotregiszter, az A, X, Y regiszterek, illetve a veremmutató értéket (ebben a sorrendben). A kiírás formátuma megfelel a ; utasítás formátumának, s ezért a regiszterek tartalma azonnal módosítható.

Példák: R

```
  PC   SR AC XR YR SP
; 8745 00 81 00 05 F6
```

S

A memória adott részének kiírása a háttértárakra.

Szintaxis: S <filenév> <egységszám> <cím1> <cím2>

A parancs a C-16 memóriájának <cím1>-től <cím2>-ig terjedő részét menti el az <egységszám>-nak megfelelő eszközre <filenév> filenév alatt. A <cím2> cím tartalma **már** nem mentődik el.

Példák: S "RAJZ" 08 2000 4000
S "KEPERNYO" 01 0400 0BFF

T

Memóriarész átmásolása.

Szintaxis: T <cím1> <cím2> <cím3>

A parancs a memória <cím1>-től <cím2>-ig terjedő részét másolja át a <cím3>-tól kezdődően. A <cím2> tartalma **már** nem kerül átmásolásra.

Példák: T 2000 2FFF 3000
T 1C00 1DFF 1E00

V

A memória tartalmának összehasonlítása egy file tartalmával.

Szintaxis: V <filenév> <egységszám>

A parancs összehasonlítja a memória tartalmát a file tartalmával. Az összehasonlítös az első eltérésnél abbamarad. Ebben az esetben ?VERIFY ERROR hibaüzenetet kapunk.

Példák: V "PROGRAM",08

X

Visszatérés a BASIC-be.

Szintaxis: X

A parancs kiadása után megint használhatjuk a BASIC programszerkesztőjét. A gépi kódú monitor parancsai pedig csak egy újabb MONITOR parancs kiadása után használhatók. A TEDMON önmaga nem módosítja a BASIC munkaterületet, ezért a program és a változók sértetlenül megmaradnak. Valamely TEDMON paranccsal természetesen tönkretehetjük a BASIC programot, illetve a változókat.

>

A memória tartalmának módosítása.

Szintaxis: > <cím> <adatok>

A <cím>-től kezdődően a memória tartalmát az <adatok>-ra cseréli át. Az <adatok> maximum 8, egymástól vesszővel elválasztott hexadecimális számot tartalmazhat.

Példák: > 0800 64 64 64 64
> 0400 03 03 03 03

;

A processzor regisztereinek tartalmát módositja.

Szintaxis: ; <cím> <státusz> <A> <X> <Y> <veremmut.>

A parancs hatása módositja a regiszterek értékét. Pontosabban arról van szó, hogy a G TEDMON parancs kiadása után a vezérlés a ; parancsban megadott <cím>-től, s az ott megadott regiszter tartalommal kezd el futni. A G paranccsal a <cím> még módosítható, de a regiszterek tartalma már nem.

FÜGGELÉK

F.1 BASIC alapszavak

A függelék felsorolja a BASIC alapszavakat, a fenntartott változókat, illetve ezek rövidítését. A 'képernyő' címszó alatt a rövidítés használatakor a képernyőn látható alakot adjuk meg; feltéve, hogy a 'kisbetűk/nagybetűk' karakterkészletet használjuk. A 'nagybetűk/grafikák' karakterkészlet használata esetén a siftelt (emelt) betűk helyén grafikus jelek láthatók. Például aB helyett A▌ jelenik meg.

Parancs	Rövidítés	Képernyő	Függvénytípus
ABS	A SHIFT B	aB	Numerikus
AND	A SHIFT N	aN	
ASC	A SHIFT S	aS	Numerikus
ATN	A SHIFT T	aT	Numerikus
AUTO	A SHIFT U	aU	
BACKUP	B SHIFT A	bA	
BOX	B SHIFT O	bO	
CHAR	CH SHIFT A	chA	
CHR$	C SHIFT H	cH	Sztring
CIRCLE	C SHIFT I	cI	
CLOSE	CL SHIFT O	clo	
CLR	C SHIFT L	cL	
CMD	C SHIFT M	cM	
COLLECT	COL SHIFT L	colL	
COLOR	CO SHIFT L	coL	
CONT	C SHIFT O	cO	
COPY	CO SHIFT P	coP	
COS	nincs	cos	Numerikus
DATA	D SHIFT A	dA	
DEC	nincs	dec	Numerikus
DEF	D SHIFT E	dE	
DELETE	DE SHIFT L	deL	
DIM	D SHIFT I	dI	
DIRECTORY	DI SHIFT R	diR	
DLOAD	D SHIFT L	dL	
DO	nincs	do	
DRAW	D SHIFT R	dR	

Parancs	Rövidítés	Képernyő	Függvénytípus
DS	nincs	ds	Numerikus
DS$	nincs	ds$	Sztring
DSAVE	D SHIFT S	dS	
EL	nincs	el	Numerikus
END	E SHIFT N	eN	
ER	nincs	er	Numerikus
ERR$	nincs	err$	Sztring
EXIT	E SHIFT X	eX	
EXP	E SHIFT X	eX	Numerikus
FN	nincs	fn	
FOR	F SHIFT O	fO	
FRE	F SHIFT R	fR	Numerikus
GET	G SHIFT E	gE	
GETKEY	GETK SHIFT E	getkE	
GET#	nincs		
GOSUB	GO SHIFT S	goS	
GOTO	G SHIFT O	gO	
GRAPHIC	G SHIFT R	gR	
GSHAPE	G SHIFT S	gS	
HEADER	HE SHIFT A	heA	
HEX$	H SHIFT E	hE	Sztring
IF	nincs	if	
INPUT	nincs	input	
INPUT#	I SHIFT N	iN	
INSTR	IN SHIFT S	inS	Numerikus
INT	nincs	int	Numerikus
JOY	J SHIFT O	jO	Numerikus
KEY	K SHIFT E	kE	
LEFT$	LE SHIFT F	leF	Sztring
LEN	nincs	len	Numerikus
LET	L SHIFT E	lE	
LIST	L SHIFT I	lI	
LOAD	L SHIFT O	lO	
LOCATE	LO SHIFT C	loC	
LOG	nincs	log	Numerikus
LOOP	LO SHIFT O	loO	
MID$	M SHIFT I	mI	Sztring
MONITOR	M SHIFT O	mO	
NEW	nincs	new	
NEXT	N SHIFT E	nE	
NOT	N SHIFT O	nO	
ON	nincs	on	
OPEN	O SHIFT P	oP	
OR	nincs	or	
PAINT	P SHIFT A	pA	
PEEK	P SHIFT E	pE	Numerikus
POKE	P SHIFT O	pO	
POS	nincs	pos	Numerikus
PRINT	?	?	

Parancs	Rövidítés	Képernyő	Függvénytípus
PRINT#	P SHIFT R	pR	
PRINTUSING	?US SHIFT I	?usI	
PUDEF	P SHIFT U	pU	
RCLR	R SHIFT C	rC	Numerikus
RDOT	R SHIFT D	rD	Numerikus
READ	R SHIFT E	rE	
REM	nincs	rem	
RENAME	RE SHIFT N	reN	
RENUMBER	REN SHIFT U	renU	
RESTORE	RE SHIFT S	reS	
RESUME	RES SHIFT U	resU	
RETURN	RE SHIFT T	reT	
RGR	R SHIFT G	rG	Numerikus
RIGHT$	R SHIFT I	rI	Sztring
RLUM	R SHIFT L	rL	Numerikus
RND	R SHIFT N	rN	Numerikus
RUN	R SHIFT U	rU	
SAVE	S SHIFT A	sA	
SCALE	SC SHIFT A	scA	
SCNCLR	S SHIFT C	sC	
SCRATCH	SC SHIFT R	scR	
SGN	S SHIFT G	sG	Numerikus
SIN	S SHIFT I	sI	Numerikus
SOUND	S SHIFT O	sO	
SPC(	S SHIFT P	sP	Speciális
SQR	S SHIFT Q	sQ	Numerikus
SSHAPE	S SHIFT S	sS	
STATUS	ST	st	Numerikus
STEP	ST SHIFT E	stE	
STOP	S SHIFT T	sT	
STR$	ST SHIFT R	stR	Sztring
SYS	S SHIFT Y	sY	
TAB(	T SHIFT A	tA	Speciális
TAN	nincs	tan	Numerikus
THEN	T SHIFT H	tH	
TIME	TI	ti	Numerikus
TIME$	TI$	ti$	Sztring
TO	nincs	to	
TRAP	T SHIFT R	tR	
TROFF	TRO SHIFT F	troF	
TRON	TR SHIFT O	trO	
UNTIL	U SHIFT N	uN	
USR	U SHIFT S	uS	Numerikus
VAL	V SHIFT A	vA	Numerikus
VERIFY	V SHIFT E	vE	
VOL	V SHIFT O	vO	
WAIT	W SHIFT A	wA	
WHILE	W SHIFT H	wH	

F.2 BASIC hibaüzenetek

Ez a függelék a C-16 BASIC interpreter által küldött hibaüzeneteket tartalmazza, illetve az azokat **előidéző okokat** ismerteti. A hibaüzenet mindig egy ? kiírásával kezdődik és az ERROR (hiba) szóval végződik. Ha a hiba programfutás közben történt, annak a sornak a száma is kiíródik, ahol az interpreter a hibát észlelte. ***Nem biztos***, hogy ténylegesen az a sor a hiba okozója.

BAD DISK (36)
Nem megfelelően megformázott lemezt használtunk.

BAD SUBSCRIPT (18)
A program egy tömb olyan elemére hivatkozott, amely túlmutat az egyáltalán lehetséges, vagy a DIM utasításban megadott határon.

BREAK (30)
A program végrehajtása a <STOP> gomb lenyomása, vagy egy STOP utasítás végrehajtása miatt félbeszakadt. Az üzenet tartalmazza annak a sornak a számát, ahol a program megállt.

CAN'T CONTINUE (26)
A CONT paranccsal nem tudjuk folytatni a programot, mert 1/ a programot még nem indítottuk el a RUN paranccsal; 2/ hiba után akarjuk továbbindítani a programot; 3/ időközben átszerkesztettük a programot.

CAN'T RESUME (31)
A RESUME utasítást a rendszer nem tudja végrehajtani, mert nem történt hiba, és nem lépett be a hibarutinba.

DEVICE NOT PRESENT (5)
A megcímzett I/O eszköz nincs a rendszerben, vagy nem üzemképes (pl. nincs bekapcsolva).

DIRECT MODE ONLY (34)
Csak parancs módban használható utasítást kíséreltünk meg programban használni.

DIVISION BY ZERO (20)
Nullával osztottunk.

EXTRA IGNORED (figyelmeztetés)
Túl sok adatot gépeltünk be az INPUT utasítás után, s csak az első néhány került elfogadásra.

FILE DATA (24)
Egy megnyitott file-ból sztring típusú adat érkezett, a program viszont számadatot várt.

FILE NOT FOUND (4)
Az OPEN vagy LOAD utasításban megadott file nem szerepel a lemez katalógusában, vagy a szalagon való keresés közben az interpreter END-OF-TAPE (szalag vége) fejlécet (header-t) talált.

FILE NOT OPEN (3)
A CLOSE, CMD, PRINT#, INPUT# vagy GET# által használni kívánt file-t még nem nyitottuk meg.

FILE OPEN (2)
Egy olyan file megnyitására történt kísérlet, amelyet már egyszer megnyitottunk, de még nem zártunk le.

FORMULA TOO COMPLEX (25)
A feldolgozás alatt levő kifejezést több részre kell bontani ahhoz, hogy a rendszer ki tudja értékelni. (Pl. a kifejezés túl sok zárójelet tartalmaz).

ILLEGAL DIRECT (21)
Az utasítás parancs (direkt) módban nem használható.

ILLEGAL DEVICE NUMBER (9)
Valamelyik használni kívánt perifériának a hardver száma nem esik az 1-255 intervallumba.

ILLEGAL QUANTITY (14)
A szám, amelyet egy függvény vagy utasítás argumentumaként használtunk, kívül esik a megengedett értékhatárokon.

LOAD (29)
A szalagról vagy lemezről való töltés az adathordozó hibája miatt megszakadt. Elsősorban szalagról való töltés esetén fordulhat elő.

LOOP NOT FOUND (32)
Az interpreter nem találja az éppen végrehajtás alatt álló DO utasításhoz tartozó LOOP utasítást. Az utasítás hiányzik a programból.

LOOP WITHOUT DO (33)
Az interpreter nem találja az éppen végrehajtás alatt álló LOOP utasításhoz tartozó DO utasítást. A program nem megfelelően strukturált, vagy nem is hajtottuk végre a DO utasítást, vagy már törlődött a rá vonatkozó hivatkozás a veremből.

MISSING FILE NAME (8)
Egy LOAD, OPEN, SAVE utasításban nem, vagy nem jól adtuk meg a file nevét.

NEXT WITHOUT FOR (10)
Next utasítást használtunk egy neki megfelelő, a végrehajtásban azt megelőző FOR utasítás nélkül. Okozhatják hibásan egymásba skatulyázott ciklusok; vagy az, ha a NEXT utáni változó név nem felel meg a FOR utasítás ciklusváltozójának (elírtuk).

NO GRAPHICS AREA (35)
A nagyfelbontású képernyőt kíséreljük meg használni, de még nem adtuk ki a megfelelő GRAPHIC utasítást.

NOT INPUT FILE (6)
Egy outputként kijelölt file-t INPUT# vagy GET# utasításban akartunk használni.

NOT OUTPUT FILE (7)
Egy inputként kijelölt file-ba akartunk PRINT# utasítással írni.

OUT OF DATA (13)
A READ utasítás nem tudott a DATA-kból adatot olvasni, mert már az összeset felhasználtuk.

OUT OF MEMORY (16)
Nincs elegendő RAM memória a program vagy a változók számára. Akkor is ezt a hibajelzést kapjuk, ha túl sok FOR ciklust skatulyáztunk egymásba, vagy a megengedettnél nagyobb mélységű a GOSUB-ok hívása.

OVERFLOW (15)
A számítási műveletek eredménye nagyobb, mint a megengedett legnagyobb szám, amely 1.70141883E+38.

REDIM'D ARRAY (19)
Egy tömböt újból definiálni akartunk, holott csak egyszer lehet. Ha egy tömbváltozó a tömb dimenzionálását megelőzően előfordul a programban, akkor automatikus tömbdeklaráció hajtódik végre, így egy ezt követő DIM utasítás már hibát eredményez!

REDO FROM START (figyelmeztetés)
Az INPUT utasítás végrehajtása során számjegy helyett karakter adatot írtunk be. Újra írjuk be az adatokat, most már helyesen, és a program futása folytatódik.

RETURN WITHOUT GOSUB (12)
RETURN utasítást hajtottunk végre azt megelőző GOSUB hívás nélkül.

STRING TOO LONG (23)
Az eredményül kapott sztring több, mint 255 karaktert tartalmaz.

SYNTAX (11)
Az utasítás nem felismerhető a C-16 BASIC számára: hiányzó zárójel, hibás kulcsszó stb.

TOO MANY FILES (1)
Egyszerre maximum 10 nyitott file lehet. Ennél többet kíséreltünk meg megnyitni.

TYPE MISMATCH (22)
Szám helyett sztringet, (vagy fordítva) használtunk.

UNDEF'D FUNCTION (27)
Felhasználó által definiált függvényre hivatkoztunk, de azt még nem definiáltuk a DEF FN utasítással.

UNDEF'D STATEMENT (17)
Nem létező sorra történt hivatkozás GOTO, GOSUB, RUN, IF, illetve ON utasításokban.

VERIFY (28)
A szalagra vagy lemezre másolt program nem egyezik meg a memóriában levővel.

F.3 DOS hibaüzenetek

A 20-nál kisebb hibakódú üzeneteket figyelmen kívül lehet hagyni. Ezek közül csak a 0 és 1 kódszámnak van közvetlen jelentése.

0 OK
Minden rendben.

1 FILES SCRATCHED
Az SCRATCH (törlés) DOS-parancs végrehajtása után kapjuk ezt az üzenetet. Az üzenet közli a törölt file-ok számát is.

20 READ ERROR
A blokk bevezető információja hiányzik. A DOS a kívánt blokk elején levő információt (header) nem tudja azonosítani. Ennek az oka egy nem létező szektorszám is lehet. Elképzelhető, hogy a 'header' megsérült.

21 READ ERROR
A blokk szinkronizációs byte-jai hiányoznak. A DOS képtelen a blokk kezdetét jelentő szinkronizációs jelet megtalálni az adott sávban (track-en). Az oka lehet az író/olvasó fej hibás mozgása, a lemez hiánya, vagy nem formázott lemez használata.

22 READ ERROR
Az adatblokk hiányzik. A DOS egy nem megfelelően felírt blokkot kísérelt meg olvasni vagy ellenőrizni. A hiba a B-R és B-W DOS parancsok esetén fordul elő.

23 READ ERROR
Ellenőrző összeg hiba. A DOS a blokkot be tudta olvasni a lemezegység pufferébe, de az ellenőrző összeg nem egyezik meg a blokkban tárolttal.

24 READ ERROR
Byte dekódolási hiba. A blokkot, vagy az előtte levő bevezető információt a DOS-nak sikerült a pufferba beolvasnia, de egy nem megengedett bitképet talált.

25 WRITE ERROR
Az írás ellenőrzése közben a DOS eltérést talált a memóriájában tárolt és a blokkba kiírt adatok közt.

26 WRITE PROTECT ON
A DOS ezt a hibaüzenetet küldi, ha egy blokk lemezre való írását kíséreltük meg, és az írást engedélyező rés a lemezen nem volt kivágva.

27 READ ERROR
A DOS a blokk bevezető információjának (header) olvasása közben ellenőrző összeg hibát észlelt.

28 WRITE ERROR
Túl hosszú adatblokk. A DOS-nak egy adatblokk írásának a befejezése után meghatározott időn belül érzékelnie kell a következő blokk szinkronizációs jelét. Ha ez nem történik meg, a fenti hibajelzést kapjuk.

29 DISK ID MISMATCH
Ez az üzenet egyaránt jelentheti, hogy a lemezt még nem formáztuk meg, vagy pedig azt, hogy a blokk bevezető információja tönkrement, s ezért a DOS nem megfelelő ID számot olvasott be.

30 SYNTAX ERROR
A 15. csatornán elküldött üzenetet a DOS képtelen értelmezni. (Például a parancsban a kelleténél több file név szerepel).

31 SYNTAX ERROR
A DOS a 15. csatornán elküldött parancsot nem tudja azonosítani. A parancsnak az első pozíción kell kezdődnie.

32 SYNTAX ERROR
A parancs hosszabb, mint 58 karakter.

33 SYNTAX ERROR
Érvénytelen file név. A SAVE vagy OPEN utasításokban nem megfelelő paraméterek szerepelnek.

34 SYNTAX ERROR
A file név hiányzik. A DOS nem találta meg a parancsban a file nevét. (Például a kettőspont hiányzik a parancsból.)

39 SYNTAX ERROR
A 15. csatornán elküldött parancs azonosíthatatlan.

50 RECORD NOT PRESENT
Az INPUT# vagy GET# utasítások használata során a relatív file utolsó rekordján túl akartunk olvasni.

51 OVERFLOW IN RECORD
A PRINT# utasítás végrehajtása során elértük a rekord határt, és így a be nem fért byte-ok elvesztek.

52 FILE TOO LARGE
Relatív file használata közben olyan nagy rekordszámot használtunk, amelyik a lemez betelését eredményezné.

60 WRITE FILE OPEN
Egy írásra megnyitott file-t, annak lezárása nélkül, olvasásra

kíséreltük meg újból megnyitni.

61 FILE NOT OPEN
Ezt az üzenetet akkor kapjuk, amikor egy, a DOS által még meg nem nyitott file-t kísérltünk meg használni.

62 FILE NOT FOUND
A file, amire hivatkoztunk egy BASIC utasításban vagy DOS parancsban, nem szerepel a lemez katalógusában.

63 FILE EXISTS
Az újonnan létrehozni kívánt file neve már szerepel a katalógusban.

64 FILE TYPE MISMATCH
A megadott file típusa nem egyezik meg a katalógusban szereplő típussal.

65 NO BLOCK
Az üzenet jelzi, hogy a B-A DOS parancs segítségével lefoglalni kívánt blokk már foglalt. A sáv- és szektorszám a legelső, nagyobb indexű szabad blokk adatait tartalmazza. Ha mindkettő 0, akkor ilyen szabad blokk már nincs.

66 ILLEGAL TRACK AND SECTOR
A DOS végrehajtása közben egy nem létező sáv-, szektorszám párosításnak megfelelő blokkot akartunk használni.

67 ILLEGAL SYSTEM T OR S
A DOS egyáltalán nem létező sávot vagy szektort próbált meg használni.

70 NO CHANNEL
A kívánt csatorna nem elérhető, vagy már az összes csatornát használjuk. A DOS egyszerre 5 szekvenciális vagy 6 közvetlen elérésű file-t tud kezelni.

71 DIRECTORY ERROR
A lemezen levő BAM nem egyezik meg a DOS memóriájában levővel. Ilyenkor az I parancs kiadásával célszerű újrainicializálni a lemezegységet.

72 DISC FULL
Nincs szabad blokk a lemezen, vagy betelt a katalógus.

73 CBM DOS V2.6 1541
A DOS 1.x és 2.x variánsai egymás lemezeit kölcsönösen olvassák, de nem írják felül. A fenti üzenet azt jelenti, hogy egy nem megfelelő verziójú DOS-szal formázott lemezre akarunk írni. Bekapcsolás után ugyanezt az üzenetet kapjuk.

74 DRIVE NOT READY
Nincs egyetlen lemez sem a lemezmeghajtóban.

F.4 Szabvány BASIC programok átalakítása a C-16 BASIC-re

Ha nem C-16 BASIC-ben írt BASIC programokkal rendelkezünk, és ezeket C-16-on akarjuk futtatni, akkor előbb néhány apróbb változtatást kell elvégeznünk. Néhány tanácsot adunk, hogy ezt az átalakítást megkönnyítsük.

Sztring típusú tömbök

A C-16 BASIC a DIM A$(I,J) utasítást úgy értelmezi, hogy egy kétdimenziós (I+1)*(J+1) méretű tömböt definiáltunk. A standard BASIC-ben ez egy egydimenziós (J elemszámú) tömböt definiál, melynek minden egyes eleme egy I hosszúságú sztring. Ezért ezeket a deklarációkat át kell írni; DIM A$(I,J)-ből például egyszerűen DIM A$(J) lesz.

A C-16 BASIC a sztringműveletekre a MID$, RIGHT$ illetve a LEFT$ függvényeket használja. A standard BASIC és más BASIC-ek is megengedik az A$(I TO J) használatát, ami az A$ sztring I-ik karakterétől J-ik karakteréig terjedő sztringet jelenti. Az A$(I TO J)=X$ értékadás ebben az esetben azt jelenti, hogy ezt a részt az X$-ra cseréljük ki. Ezt az utasítást C-16-on így valósítjuk meg:

```
A$=LEFT$(A$,I-1)+X$+MID$(A$,J+1)
```

Többszörös értékadás

Néhány BASIC interpreter megengedi a 10 LET B=C=0 típusú értékadást, melynek hatására B és C értéke is 0 lesz. A C-16-on ilyen értékadás nincs. A fenti utasítást a gép LET B=(C=0)-nak értelmezi, és attól függően, hogy C nulla volt-e vagy sem, B értéke -1, illetve 0 lesz. A kívánt hatást a 10 B=0:C=0 utasítással érhetjük el.

Több utasításból álló sorok

Nem mindegyik BASIC interpreter használja a kettőspontot az egy sorban levő utasítások elválasztására. Ezeket a C-16-on természetesen kettősponttal kell elválasztani.

MAT függvények

Elsősorban minigépeken futó BASIC interpreterek lehetővé teszik mátrix müveletek használatát. Ezeket C-16-on külön alprogramok megírásával lehet csak előállítani.

F.5 Képernyő kódok

Az alábbi táblázatban megtalálhatjuk a C-16 teljes karakterkészletét. A táblázat megadja, hogy milyen kódot kell a POKE utasítással a képernyő memóriába beírnunk ahhoz, hogy a kívánt karakter jelenjen meg. A PEEK utasítással kiolvasott adatokról megállapíthatjuk, hogy mely karakternek felelnek meg. Két karakterkészlet áll rendelkezésünkre, azonban egyidejüleg csak az egyiket használhatjuk. A karakterkészletek átváltása a SHIFT és C= billentyűk egyidejű lenyomásával történik. A BASIC-ből PRINT CHR$(14), illetve PRINT CHR$(142) utasításokkal állíthatjuk be a kívánt karakterkészletet. Bármelyik betű, karakter vagy grafikus jel inverz alakban is megjeleníthető. Az inverz karaktereket a táblázatban szereplő értékek 128-cal való megnövelésével kapjuk. A következő intervallumba eső kódszámú karakterek a két karakterkészletben azonosak:

27-64,91-93,96-104,106-121,123-127.

Képernyő kódok:

@	0		32
A a	1	!	33
B b	2	"	34
C c	3	#	35
D d	4	$	36
E e	5	%	37
F f	6	&	38
G g	7	.	39
H h	8	(	40
I i	9	)	41
J j	10	*	42
K k	11	+	43
L l	12	,	44
M m	13	-	45
N n	14	'	46
O o	15	/	47
P p	16	0	48
Q q	17	1	49
R r	18	2	50
S s	19	3	51
T t	20	4	52
U u	21	5	53
V v	22	6	54
W w	23	7	55
X x	24	8	56
Y y	25	9	57
Z z	26	:	58
[	27	;	59
<font>	28	<	60
]	29	=	61
^	30	>	62
vissza-nyíl	31	?	63

Képernyő kódok (folytatás):

Jel	Karakter	Kód	Karakter	Kód
		64	szóköz	96
	A	65		97
	B	66		98
	C	67		99
	D	68		100
	E	69		101
	F	70		102
	G	71		103
	H	72		104
	I	73		105
	J	74		106
	K	75		107
	L	76		108
	M	77		109
	N	78		110
	O	79		111
	P	80		112
	Q	81		113
	R	82		114
	S	83		115
	T	84		116
	U	85		117
	V	86		118
	W	87		119
	X	88		120
	Y	89		121
	Z	90		122
		91		123
		92		124
		93		125
π		94		126
		95		127

F.6 ASCII kódok

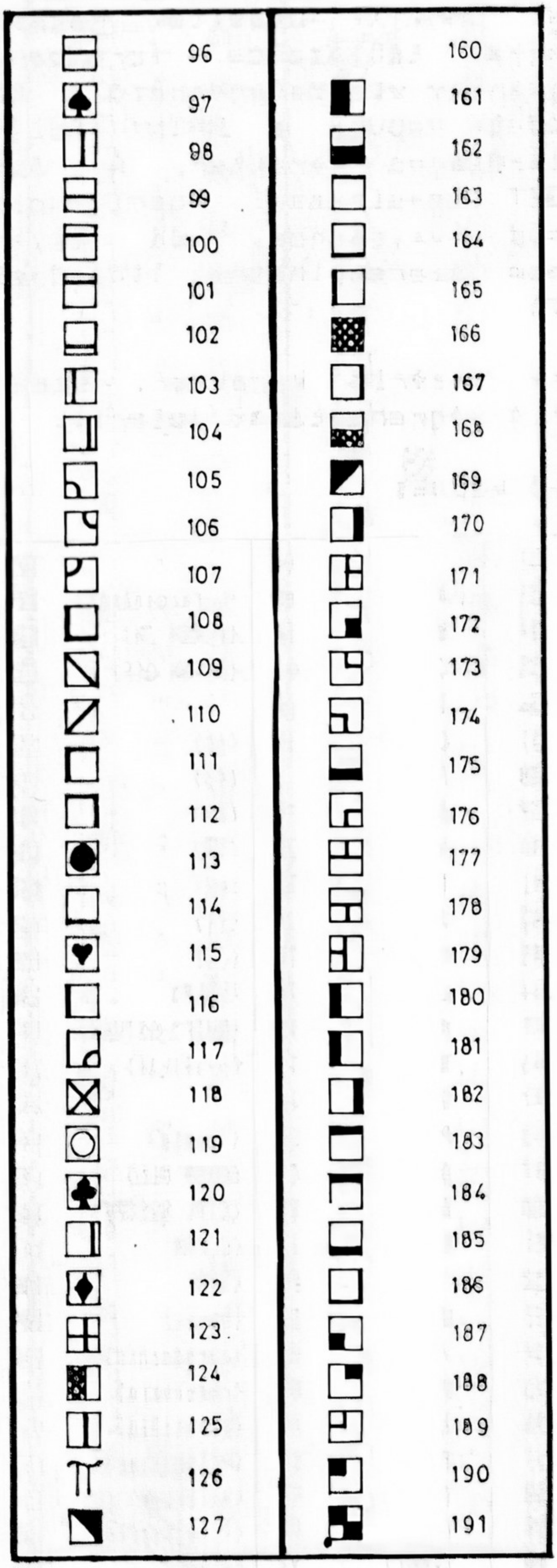
Ez a függelék a PRINT CHR$(X) utasítás hatására megjelenő karaktereket tartalmazza táblázatos formában, X összes lehetséges értékére. Ugyanígy visszakereshető a táblázatból az is, hogy milyen számkódot kapunk a PRINT ASC("S") utasítás eredményéül, ha S tetszőleges karakter. A táblázat hasznos segítséget nyújt a GET utasítással beolvasott karakterek kiértékeléséhez, üzemmód váltáshoz, és olyan karakterek kiírásához, amelyek nem szerepelhetnek idézőjelben (pl. a karakterkészlet kapcsoló.)

A karakterek egy része vezérlő karakter. Ezek 'kiírása' a megfelelő vezérlő funkció végrehajtását jelenti.

Alfanumerikus és vezérlő kódok:

	0	<szóköz>	32		64		128
	1	!	33	A	65	<narancssárga>	129
	2	"	34	B	66	<FLASH ON>	130
	3	#	35	C	67	<FLASH OFF>	131
	4	$	36	D	68		132
<fehér>	5	%	37	E	69	<f1>	133
	6	&	38	F	70	<f3>	134
	7	'	39	G	71	<f5>	135
<C=-CTRL tiltás>	8	(	40	H	72	<f7>	136
<C=-CTRL engedély>	9	)	41	I	73	<f2>	137
	10	*	42	J	74	<f4>	138
	11	+	43	K	75	<f6>	139
	12	,	44	L	76	<HELP>	140
<RETURN>	13	-	45	M	77	<SHIFT-RETURN>	141
<betűk>	14	.	46	N	78	<grafikák>	142
	15	/	47	O	79		143
	16	0	48	P	80	<fekete>	144
<CRSR LE>	17	1	49	Q	81	<CRSR FEL>	145
<CTRL RVSON>	18	2	50	R	82	<CTRL RVSOFF>	146
<HOME>	19	3	51	S	83	<CLEAR>	147
<DEL>	20	4	52	T	84	<INST>	148
	21	5	53	U	85	<barna>	149
	22	6	54	V	86	<sárgászöld>	150
	23	7	55	W	87	<rózsaszín>	151
	24	8	56	X	88	<kékeszöld>	152
	25	9	57	Y	89	<világoskék>	153
	26	:	58	Z	90	<sötétkék>	154
<ESC>	27	;	59	[	91	<világoszöld>	155
<piros>	28	<	60	<font>	92	<bíbor>	156
<CRSR jobbra>	29	=	61	]	93	<CRSR balra>	157
<zöld>	30	>	62	^	94	<sárga>	158
<kék>	31	?	63	<visszanyíl>	95	<encián>	159

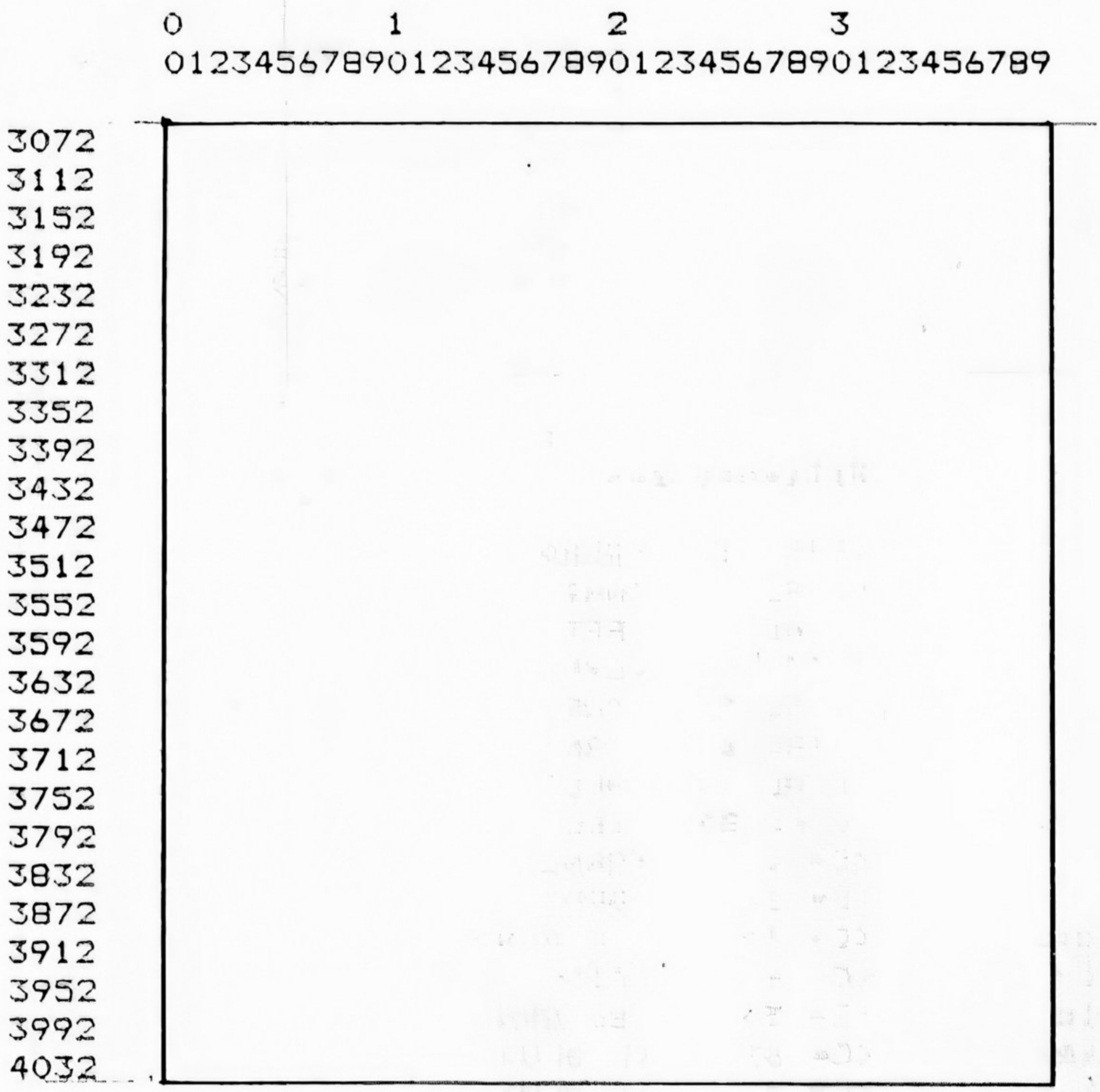
ASCII kódok (folytatás: grafikus jelek):



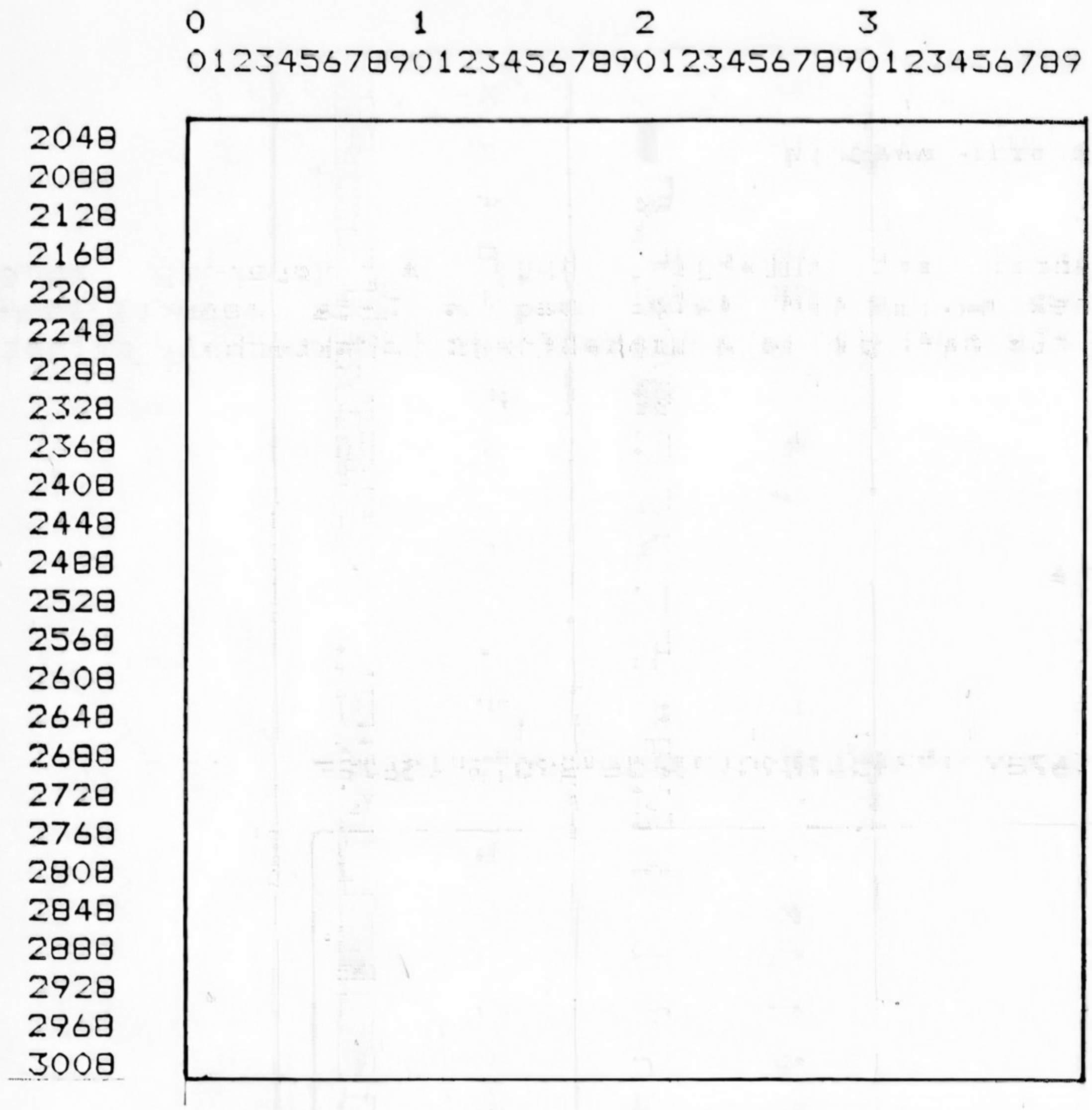
F.7 Képernyő és szin memória

A következő ábrák azt mutatják, hogy a képernyő egyes karakterhelyeinek melyik cím felel meg a C-16 memóriájában, illetve melyik cím definiálja a szóbanforgó karakterhely színét.

Képernyő memória



Szín memória



Kód	Szín	Billentyűzés	
0	fekete	<CTRL 1>	<BLK>
1	fehér	<CTRL 2>	<WHT>
2	piros	<CTRL 3>	<RED>
3	encián	<CTRL 4>	<CYN>
4	bíbor	<CTRL 5>	<PUR>
5	zöld	<CTRL 6>	<GRN>
6	kék	<CTRL 7>	<BLU>
7	sárga	<CTRL 8>	<YEL>
8	narancs	<C= 1>	<ORNG>
9	barna	<C= 2>	<BRN>
10	sárgászöld	<C= 3>	<YL GRN>
11	rózsaszín	<C= 4	<PINK>
12	kékeszöld	<C= 5>	<BL GRN>
13	világoskék	<C= 6>	<L BLU>
14	sötétkék	<C= 7>	<D BLU>
15	világoszöld	<C= 8>	<L GRN>

F.8 Frekvencia értékek

Táblázatunk azt mutatja meg, hogy egy adott hang megszólaltatásához milyen értéket kell a SOUND utasításban használni.

Hang	SOUND	Frekvencia
1 a	7	110
2 h	118	123.5
3 c	169	130.8
4 d	262	146.8
5 e	345	164.7
6 f	383	174.5
7 g	453	195.9
8 a	516	220.2
9 h	571	246.9
10 c	596	261.4
11 d	643	293.6
12 e	685	330
13 f	704	349.6
14 g	739	392.5
15 a	770	440.4
16 h	798	494.9
17 c	810	522.7
18 d	834	588.7
19 e	854	658
20 f	864	699
21 g	881	782.2
22 a	897	880.7
23 h	911	989.9
24 c	917	1045
25 d	929	1177
26 e	939	1316
27 f	944	1398
28 g	953	1575

F.9 Memória térkép

Cím		Tartalom
0- 2047	$0000-$07FF	Rendszerváltozók
2048- 3071	$0800-$0BFF	Színmemória
3072- 4095	$0C00-$0FFF	Képernyő memória
4096-16383	$1000-$3FFF	BASIC munkaterület
6144- 7167	$1800-$1BFF	Nagyfelbontású képernyő (fényerő)
7168- 8191	$1C00-$1FFF	Nagyfelbontású képernyő (szín)
8192-16383	$2000-$3FFF	Bittérkép
32768-53247	$8000-$CFFF	BASIC ROM
53248-55295	$D000-$D7FF	Karakter generátor ROM
55296-65535	$D800-$FFFF	KERNAL ROM
64768-65529	$FD00-$FF80	I/O RAM

Utószó a második kiadáshoz

A C-16 és C-116 BASIC felhasználói kézikönyv első kiadása igen szerencsés pillanatban került az olvasók kezébe: akkor amikor ezek a gépek az üzletekben. Ennek is köszönhetően már a második kiadást készítjük elő. Ennek utószavában néhány kiegészítést teszünk, illetve kijavítjuk az általunk felfedezett hibákat.

Külön felhívjuk a figyelmet a C-16 és C-116 BASIC azon sajátosságárá, hogy feltételes utasításokban használhatjuk az ELSE alapszót is.

Az utasítás szintaktikája a következő:

IF <feltétel> THEN <utasitássorozat1> : ELSE <utasitássorozat2>

Ha a feltétel a kiértékelése után igaznak bizonyul, akkor az első, ha hamisnak, a második utasitássorozat hajtódik végre. Az ELSE a Commodore személyi számítógépek egyik legbosszantobb hibáját küszöböli ki. (Igaz nem teljes egészében, hiszen az utasításnak az ELSE résszel együtt **egyetlen** sorban el kell férnie.)

Példák:

```
(i)   IF X>0 THEN X$="POZITIV": ELSE X$="NEM POZITIV"
(ii)  IF DS=0 THEN RETURN: ELSE GOTO 1200
(iii) IF X>=0 THEN ?SQR(X): ELSE ?"NINCS GYOKE!"
```

A LOCATE utasitás a 123. oldalon megadott szintaxistól eltérő módon is használható, és ebben az esetben az aritmetikai kifejezések értékei **a pont polárkoordinátáit** adják meg:

LOCATE <arit.kif.1>**;**<arit.kif.2>

(Az eltérés a vastagon szedett ';' az eredeti ',' helyett!).

Egy kézikönyvnél elsőrendűen fontos, hogy ne tartalmazzon sajtóhibát. Minden alkalommal a lehető legalaposabban átnézzük a kiadvány szövegét, de sajnos így is előfordul, hogy maradnak benne hibák. S persze a legzavaróbb helyen. Így például kedvenc teknősbéka programjaimnál. A 241. oldal alján levő táblázat

néhány belépési pontot rosszul tartalmaz. Ezek helyessen a következők:

Funkció	Belépési pont	Paraméter
LEFT	70	SZOG
RIGHT	80	SZOG
FORWARD	90	HOSSZ

Ennek megfelelően módosítani kell a hozzá tartozó programokat is:

SOKSZOG (243. oldal) 270. sora helyessen:

```
270 GOSUB 90: GOSUB 70
```

CSILLAG (243. oldal) 270. sora helyessen:

```
270 GOSUB 90: GOSUB 70
```

SPIRAL (244. oldal) 260-270. sorai helyessen:

```
260 DO: GOSUB 70
270 HO=HO+NOV: GOSUB 90
```

A 56. oldalon szereplő ábrán a feliratok felcserélődtek: a szög értéke arc tg x, az oldal menti felirat pedig x.

Oldal Sor Helyes szöveg

C-16 & C-116: Utószó U/3. oldal

Oldal Sor Helyes szöveg

Oldal	Sor	Helyes szöveg
11	a 12	ilyen a szerkesztője. A Sinclair gépek a képernyő utolsó két
19	a 8	addig, amíg a képernyőn vannak. Ha már nem akarunk több villogó
36	f 23	vagy pontosvesszőre végződik, az utasitás egy CHR$(13) jelet is
47	f 11	"ABRAKA"+" DABRA"="ABRAKADABRA"
66	f 2	Röviditės: cL Token:$9C(156)
69	a 10	Szintaxis: COLOR <szintipus>,<szin> [, <fényerő>]
74	a 4	20 PRINT X,HEX$(X),DEC(HEX$(X)) : X=X+1
76	f 2	40 PRINT I, FN F(I)
76	f 4	(ii) 300 DEF FN DEEK(X)=PEEK(X)+256*PEEK(X+1)
90	f 13	(v) 50 NT=NE*EXP(-B)*EXP(-K*T)
94	f 15	X=FRE(0): DIM X%(278): PRINT X-FRE(0)
97	f 3	Mód: csak program módban használható.
107	f 2	Röviditės: nincs Token:$8B (139)
135	f 19	a képernyőre a 239. memória tartalmát. Ez általában 0, lévén,
142	f 2	Röviditės: ?usI Token: USING $F3(243)
164	f 16	(i) SOUND 3,800,6000
164	a 5	1024-M
171	a 12	(iv) 5000 TR=TI
186	a 9	megnyomásának megfelelő tevékenységet, s végrehajtja a képernyő
218	f 4	33 PRINT"<CLEAR><CRSR LE> POKE-KAL VISSZA!"
222	f 4	szolgál 40-90 ciklus. A szirénahatást egy magas és egy mélyebb
235	a 15	120 GSHAPE X$,X-10,X/2-5,4
238	a 10	Egy kicsit részletesebben a teknősbéka geometria használata a
239	f 4	ezeknek egy tömöritett változatát is, ami lényegesen több szabad
242	a 9	70-nél, majd elforditjuk 120 fokkal, és ezt még kétszer

(a= alúlról, f=felülről számitott sorszám. A szélesen szedett betűk a javitottak.)

```
100 REM ***********
105 REM * ZONGORA *
110 REM ***********
115 :
120 REM KLAVIATURA KIRAJZOLASA
125 PRINT"████████"
130 PRINT"          █ █ █ | █ █ █ | █ █ | █ █ "
135 PRINT"          █ █ █ | █ █ █ | █ █ | █ █ "
140 PRINT"          █ █ █ | █ █ █ | █ █ | █ █ "
145 PRINT"          █ | | | | | | | | | | | | "
150 PRINT"          █Q |W |E |R |T |Y |U |I |O |P |@ |+ |-"
170 :
175 REM FREKVENCIAK BETOLTESE
180 DIMF(14):DIMK(255)
185 FORI=1TO14: READ F(I): NEXT I
195 :
200 REM KEZDOERTEK
205 K$="QWERTYUIOP@+-█"
210 FORI=1TOLEN(K$):K(ASC(MID$(K$,I)))=I:NEXT
215 VOL 7
220 :
250 REM A BILLENTYUZET OLVASASA
255 GETKEY A$
260 FR=K(ASC(A$))
265 IF FR=0 THEN GOTO 255
270 :
275 REM EGY HANG MEGSZOLALTATASA
300 SOUND 2,F(FR),6
305 GOTO255
306 DATA 169,262,345,383,453
310 DATA 515,571,596,643,685,704,733
320 DATA 770,798
```

```
1 L$="▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓"
2 K$="▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓"
8 KK=3072: NK=8192
10 PRINT"■■";CHR$(8);
20 N=21:M=24
30 FOR I=1 TO N
40 FOR J=1 TO M:PRINT" ";:NEXT J
50 PRINT"■"
60 NEXT I
70 FOR J=1 TO M:PRINT"■";: NEXT J: PRINT"■■";
80 I=1:J=1:PRINT"■?■■";
90 GET A$: IF A$="" THEN GOTO90
100 IF A$="." THEN GOTO 300
110 IF A$=" " THEN GOTO 330
112 HE=KK+(I-1)*40+(J-1)
125 J$=" ": IF (PEEK(HE)AND127)=42 THEN J$="*"
130 IF A$="□" THEN I=I-1: GOTO 170
140 IF A$="■" THEN I=I+1: GOTO 170
150 IF A$="■" THEN J=J+1: GOTO 170
160 IF A$="■" THEN J=J-1: GOTO 170
162 IF A$="V" THEN GOTO 800
164 IF A$=CHR$(13) THEN I=I+1:J=1:GOTO 170
165 IF A$="■" THEN GOTO 500
166 IF A$="■" THEN I=1:J=1: GOTO 170
170 IF I>N THEN I=1
171 IF I=0 THEN I=N
172 IF J>M THEN J=1
173 IF J=0 THEN J=M
179 PRINTJ$;
180 PRINT"■";:IF I=1 THEN GOTO 200
190 PRINT MID$(L$,1,I-1);
200 IF J=1 THEN GOTO 220
210 PRINT MID$(K$,1,J-1);
220 HE=KK+(I-1)*40+(J-1)
221 J$="■?■■": IF (PEEK(HE)AND127)=42 THEN J$="■*■■"
230 PRINTJ$;:GOTO 90
300 J$="*":J=J+1:IF J>M THEN J=1
310 GOTO 170
330 J$=" ":J=J+1:IF J>M THEN J=1
340 GOTO 170
500 PRINT"■";: FOR II=1 TO N
510 FOR JJ=1 TO M
512 HE=KK+(II-1)*40+(JJ-1)
520 J$="*": IF (PEEK(HE)AND127)=42 THEN J$=" "
540 PRINTJ$;: NEXT JJ: PRINT: NEXT II
550 GOTO 180
800 GRAPHIC 1,1
310 FOR KI=1 TO N
320 FOR KJ=1 TO M
830 SR=(KI-1)AND7:OS=INT((KJ-1)/8)
840 OK=(KJ-1)AND7:BI=7-OK
845 HE=NK+320*INT((KI-1)/8)+8*OS+SR
846 IF (PEEK(KK+(KI-1)*40+KJ-1)AND 127)<>42 THEN GOTO 860
850 POKE HE,PEEK(HE) OR (2↑BI)
860 NEXT KJ: NEXT KI
861 SSHAPE KE$,0,0,23,20
863 FOR KI=1 TO 7
864 FOR KJ=1 TO 10
865 IF (KI-1)*10+KJ=68 THEN KJ=10:GOTO 867
866 CHAR 1,4*(KJ-1),6+KI,RIGHT$("    "+STR$(ASC(MID$(KE$,(KI-1)*10+KJ,1))),4)
867 NEXT KJ: NEXT KI
870 POKE 239,0
880 GETKEY A$: GRAPHIC 0: GOTO 90
```

BILLENTYUZES	ASCII	MEGJELENES IDEZOJEL MODBAN
<CLEAR>	147	
<HOME>	19	
<CRSR LE>	17	
<CRSR FEL>	145	
<CRSR JOBBRA>	29	
<CRSR BALRA>	157	
<RVS ON>	18	
<RVS OFF>	146	

C-16 & C-116: különbségek

A két számítógép tulajdonképpen csak a 'kiszerelésben' tér el egymástól. Ez azt jelenti, hogy a két gép ugyanazokból az integralt áramkörökből, ugyanazon csatlakozókból, s ugyanazon ROM-ban tárolt programból épül fel, csupán ezek egyetlen kártyán való összeszerelése más. A C-116 sokkal kompaktabb, kisebb, inkább a Sinclair gépekhez hasonló megjelenésű, míg a C-16 megjelenésre teljesen azonos a VC-20, illetve a C-64 gépekkel.

A C-116 további lényeges különbsége a C-16-hoz képest a billentyűzet kiképzése. A Commodore gépek eddigi igényes billentyűzetét, egy kevésbé állóképesre cserélték át a C-116-on. Éppen ezért rendszeres iskolai használatra inkább a C-16 változatot javasoljuk.

A billentyűzet technikai megváltoztatását a mintegy feleakkora méret indokolja. Ezen túlmenően további - lényeges - változtatásokat is végrehajtottak a billentyűzeten. A Commodore gépek egyik legtöbbet bírált tulajdonsága a nem professzionális elrendezésű billentyűzet volt. Különösen a kurzor vezérlő billentyűk szokatlan helye okozott problémát. Már a C-16-on is négy különböző billentyű szolgál a kurzor mozgatására (szemben a korábbi Commodore gépek két billentyűjével). A C-116-on jobb oldalán egyetlen speciális billentyű található, amelyik a kurzor vezérlésére szolgál. A billentyű sarkainak a lenyomása felel meg az egyes kurzor vezérlő billentyűk lenyomásának.

További különbség, hogy a C-64-en egy külön tömbben található ún. funkció billentyűket a billentyűzet felső sorában helyezték el, s kiemelték a <HOME>, illetve a <DEL> billentyűt is.

A majdnem fele méretű billentyűzet azt eredményezi, hogy a billentyűkre nem lehet három jelet ráírni. Ezért a hármas jelentésű billentyűk esetén a 2. és 3. jelentésnek megfelelő karaktereket az egyes billentyűk fölé az alaplapra festették. A C-16-on ez a billentyűk felénk eső oldalán látható.

A C-16, illetve a C-116 számítógép bővítési lehetőségei, a hozzájuk kapcsolható perifériák teljes egészében megegyeznek. Azonos a két számítógép BASIC interpretere, illetve annak gépi kódú megvalósítása.

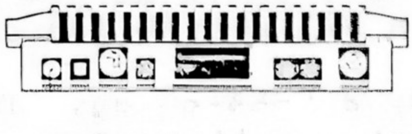
A C-116 mérete, kivitelezése miatt elsősorban otthoni tanulásra, szórakozásra való. A billentyűzettel még így is finomabban célszerű bánni, mint C-16-os társáéval, mert elsősorban a speciális kurzor mozgató billentyű könnyebben tönkremehet.

A C-116 felépítése

A C-116 számítógép egyetlen dobozba összeszerelve tartalmazza magát a számítógépet, a billentyűzetet, illetve a perifériák - köztük a televízió - illesztéséhez szükséges eszközöket. Az alapgép dobozában megtalálható a TV összekötő kábele, illetve a transzformátor. A C-116 kisebb térfogata miatt gyorsabban melegszik, mint a C-16, ezért időről időre tapintsuk meg, hogy mennyire meleg, s ha túl forrónak találjuk, akkor kapcsoljuk ki egy kis időre.

A C-116 üzembeállításának lépései megegyeznek a C-16-éval. Most ezért csak az egyes csatlakozók elhelyezkedését ismertetjük, mert az teljesen eltér a C-16-étól.

A gép jobb oldalán található a hálózati kapcsoló. **OFF** a kikapcsolt, **ON** a bekapcsolt állapotot jelzi. A gép bal oldalán **RF** felirattal található a TV kábelének a csatlakozója. A további csatlakozók a gép hátoldalán találhatók.



Mint a fenti ábrán is látható a gép hátoldalán összesen hét csatlakozó és egy nyomógomb található. Ezek balról jobbra haladva a következők:

1./ Hálózati csatlakozó (**POWER**). Ide kell a transzformátort csatlakoztatni.

2./ RESET gomb (**RESET**). A RESET gomb megnyomása újraindítja a gépet. Az esetleg tárolt BASIC program elvész, bár a memóriát nem törli a rendszer. Elsősorban olyankor célszerű használni, ha egy gépi kódú program kipróbálása közben elszáll a rendszer.

3./ Soros csatlakozó (**SERIAL**). Ide kell a lemezegységet, illetve a nyomtatót csatlakoztatni.

4./ Kazettás egység csatlakozója (**CASSETTE**). Ide kell a kazettás egységet csatlakoztatni. A C-16, illetve a C-116 gépekhez speciális magnót kell vásárolni. Nem jó sem a normális háztartási magnetofon, sem a VC-20-hoz, illetve a C-64-hez használható kazettás egység!

5./ Memória bővítő (**MEMORY EXPANSION**). Ebbe a legtöbb érintkezős csatlakozóba kell a memória bővítéseket, illetve a gyárilag ROM-ba égetett programokat tartalmazó ún. cartridge-okat behelyezni. Csak a C-116-hoz készített bővítéseket használjunk, ellenkező esetben a gép tönkremehet!

6.,7./ Botkormányok csatlakozói (**JOY1, JOY2**). Ide kell az esetleg használt botkormányokat csatlakoztatni. A C-64-hez használható botkormányok nem megfelelőek, speciálisakat kell vásárolni.

8./ Video bemenet (**VIDEO**). Ha monitort használunk TV helyett, akkor a TV kábel helyett monitor kábelt kell használnunk, amit ide kell csatlakoztatnunk.

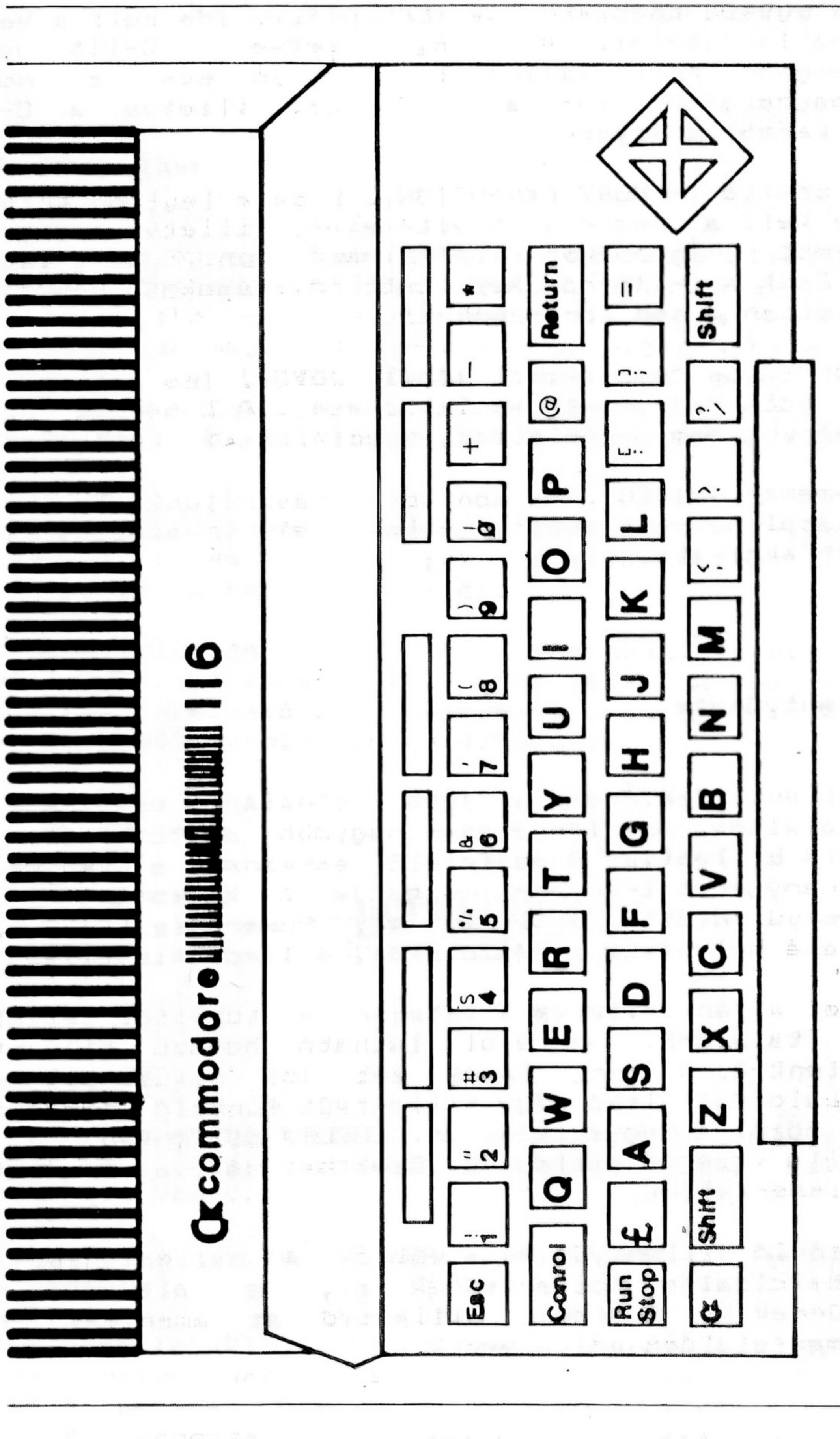
A C-116 billentyűzete

A C-116 billenttyűzetének a jobb oldalán egy speciális billentyűt találunk, ami lényegesen nagyobb a többinél. Ez a **kurzor vezérlő billentyű.** A megfelelő sarkának a lenyomása a billentyűre rányomott irányban mozgatja a kurzort. A kurzor vezérlő billenyű fölött található egy **Power** feliratú lámpa. Amikor áram alá helyezzük a készüléket, a lámpa kigyullad.

A billentyűzet alján, illetve a tetején a többitől elkülönült billentyűket találunk. Az alúl látható hosszú billentyű a **<szóköz> billentyű.** A fent levők két jól elkülönült részre oszlanak. A baloldalt levő négy billentyűt **funkció billentyűnek** hívjuk. Ezek közül a negyedik az ún. **<HELP> billentyű.** A másik kettő speciális vezérlő billentyű. Ezek hatását a későbbiekben részletesen ismertetjük.

A további vezérlő billentyűk és a váltók a billentyűzet jobb, illetve a baloldalán helyezkednek el, s általában csak méretükkel tűnnek ki. A többi billentyű az amerikai írógép-szabványnak megfelelően helyezkedik el.

A C-116 billentyűzete



Commodore 116 billentyűzet

A Commodore Plus/4

egy igazi házi számítógép, mely közeli rokonságban áll a kézikönyvben részletesen ismertetett C-16 (és C-116) típusokkal.

E fejezet célja, hogy megismertessük a felhasználókat ennek az igen kedvelt és hazánkban is egyre terjedő személyi számítógépnek a sajátosságaival, különös tekintettel a kisebb memóriájú elődöktől való eltérésekre.

1983-ban a nyugati számítógépes szaksajtóban olvashattuk, hogy a Commodore 64 egy szupergép, mert példátlan robbanást hozott a személyi számítógép piac ár/teljesítmény viszonyaiban. A 'Commodore 16 egy kész csoda' - írta egy hazai szerző két éve megjelent kötetében. A Plus/4 pedig sok tekintetben még ezt is felülmúlja...

A felhasználót legjobban érdeklő kérdések közül érdemes kiemelni a szoftverkompatibilitást.

- A C-16-ra ill. memóriabővítővel ellátott C-16-ra írt programok szinte minden esetben gond nélkül futnak a Plus/4-en. A problémát jelentő néhány kivételre ill. a futtathatóság megoldására a későbbiekben kitérünk.

- A Plus/4-re írt bármilyen szoftver működik a 64 K RAM-ra kibővített C-16 gépen.

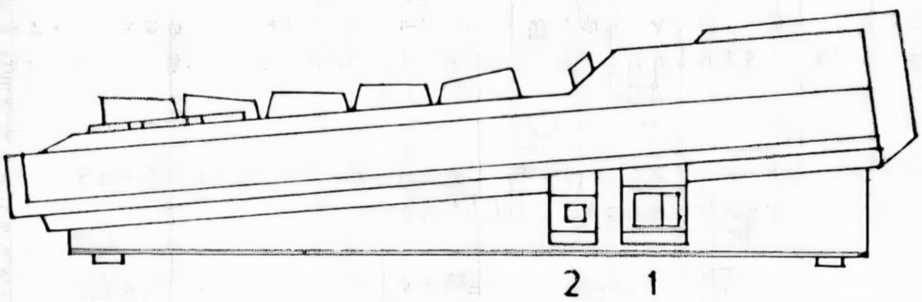
- Sokan abban a tévhitben élnek, hogy a C-64-re írt programok működnek a Plus/4-en, és ha gond van egyszerű átalakítással megoldhatjuk a problémát. Sajnos ez csak részben igaz. A BASIC programokat e kézikönyvet részletesen tanulmányozva és összehasonlítva a C-64 BASIC programozási lehetőségeivel kis gyakorlattal könnyedén átírogathatjuk egyik géptípusról a másikra. Merőben más a helyzet a gépi kódú programok esetében. Ezek átírása a két géptípus hardver felépítésbeni különbségei miatt komoly nehézségekbe ütközik. Mindezek ellenére egyetlen Plus/4 felhasználó sem keseredjen el, mert a szoftverkészítők egyre több C-64-re már korábban megjelent programot a Plus/4-re is kifejlesztenek. Ma már olyan gyakorlat alakult ki a meghatározó szoftverházaknál, hogy a két típusra párhuzamosan fejlesztik ki programjaik egy jelentős részét.

A továbbiakban csak a Plus/4 és a harmadik generációs Commodore gépcsalád többi tagjának (C-16, C-116) összehasonlító elemzésére szorítkozunk.

KAPCSOLÓK, CSATLAKOZÓK

A Plus/4 külső kialakitása ugyanolyan, mint a C-116-é, de korszerűbb és sok tekintetben célszerűbb, mint a C-16-os gépé, bár sok amatör jobban kedveli a régi nagyobb dobozokat (C-16) a hardver átalakitások, bővitések szélesebb körű alkalmazhatósága miatt. A Plus/4-hez csatlakoztatható perifériák választéka is bővebb, mint közvetlen elődjééeié, ugyanis a gyártó user-port kialakitásával növelte a felhasználói lehetőségeket.

A Plus/4 jobb oldala:



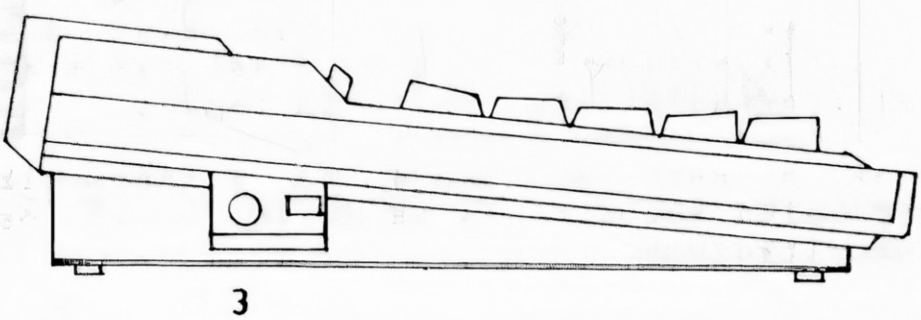
1. Üzemi kapcsoló

A gép bekapcsolt állapotáról a billentyűzet alatt bal oldalon levő piros fény ad tájékoztatást.

2. Reset kapcsoló

Segitségével ugyanúgy, mint a C-16 esetében történik a BASIC 'hideg', a RUN/STOP billentyű egyidejű megnyomásakor a BASIC programok 'meleg' inditása.

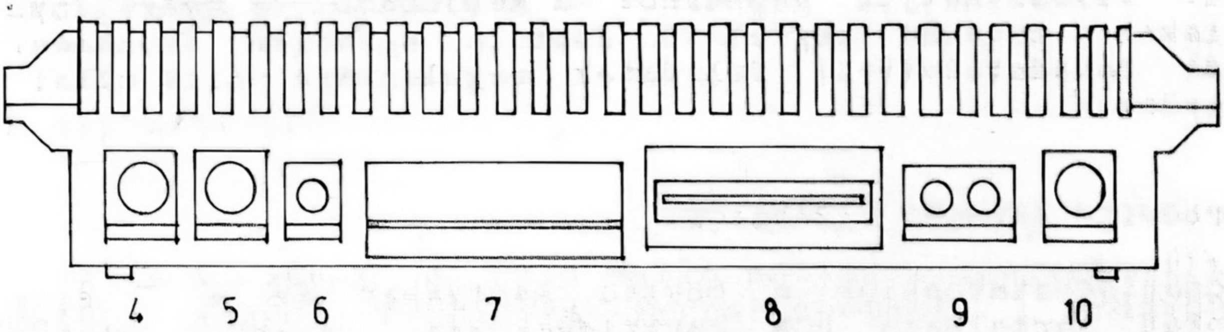
A Plus/4 bal oldala:



3. RF aljzat:

Az ide csatlakoztatott TV kábelen át jut a nagyfrekvenciás kép-hang jel a televízió készülék antenna bemenetére. Mint bármelyik más Commodore gép a Plus/4 is csak PAL rendszerben is működő készülékeken szolgáltat színes képet és hangot. A csak SECAM rendszerű színes TV a hangot csak a számítógépen szakember által történő kis mértékű beavatkozás után tudja előállítani, színes képre azonban ne számítsunk ekkor sem.

A Plus/4 hátoldala:



4. Hálózati csatlakozó (POWER)

Ide kell a transzformátort csatlakoztatni. A Plus/4 tápegységének teljesítménye jóval nagyobb, mint a C-16-é, amely csak a memóriabővítő nélküli gépek esetében kielégítő. A Plus/4 gép tápegységének teljesítményfelvétele 46VA, míg a C-16-é csak 14 VA. Így a tápegység a gép huzamosabb ideig történő használata esetén sem melegszik fel túlzott mértékben.

Az eddigi tapasztalatunk alapján a tápegység csatlakozója kétféle kivitelben készült. Az egyik egy szögletes speciális, a másik - a szerencsésebb megoldás - normál DIN szabványú csatlakozó. Ez utóbbi dugóval szerelt változat teljesen csereszabatos a C-64 gép tápegységével.

5. Soros csatlakozó aljzat (SERIAL)

Ide illeszthetjük a soros csatlakozójú lemezegységet és nyomtatót ugyanúgy, mint a C-16 ill. C-116 esetében.

6. Kazettás egység csatlakozója (CASSETTE)

Ide csatlakoztathatjuk a kazettás egységet (Datasette). A magnetofon mini DIN típusú dugón keresztül csatlakoztatható a számítógéphez. Ez is megegyezik a C-16-tal. A C-64-hez használatos magnó műszaki adatai megegyeznek a Plus/4-hez legmegfelelőbb C=1531 típuséval, de csatlakoztatása csak egy adapteren keresztül lehetséges.

7. User-port (RS-232)

Ide illeszthetjük az RS-232 interface-t. Ez a felhasználói kapu nem volt kialakítva sem a C-16-on, sem a C-116-on. Új lehetőségek nyílnak meg ezen a csatlakozási lehetőségen keresztül a Plus/4 felhasználók előtt. Az RS232 interface-en keresztül illeszthetjük gépünkhöz a nem Commodore gyártmányú perifériákat, továbbá megfelelő csatoló egységen (modemen) keresztül távadatátviteli feladatok megoldására válik alkalmassá gépünk.

8. Tárbővítő (MEMORY EXPANSION)

Ide csatlakoztathatjuk a bővítő kártyákat és a kész programokat tartalmazó ROM cartridge-okat, ugyanúgy, mint a C-16 esetében. Fontos, hogy ezeket a kártyákat csak a gép kikapcsolt állapotában illesszük be, vagy húzzuk ki.

A Commodore cég a harmadik generációs gépcsaládjához (C-16, C-116, Plus/4) kifejlesztett egy lemezmeghajtó egységet a VC 1551-et. Külsőre csak színében és csatlakozójában tér el a közismertebb és elterjedt VC 1541 típustól. Működési elve alapvetően újszerű. Ez a drive párhuzamos illesztésű. Az adatok nyolc vonalon egyidejűleg mozognak, ezáltal az adatátvitel sebessége bármiféle turbó használata nélkül mintegy ötszöröse a VC 1541 típusénak. Az illesztő kialakítása olyan, hogy a memóriabővítő kapura csatlakoztatva az illesztőn keresztül másik meghajtó egység vagy cartridge is csatlakoztatható a géphez.

Az itt leírt 1551-es lemezmeghajtó legfőbb hátránya, hogy kizárólag a C-16, C-116 vagy Plus/4 gépekhez csatlakoztatható.

9. Botkormányok csatlakozói (JOY)

Ezek az aljzatok is speciális mini DIN szabványúak. Ilyen kimenetű botkormányok beszerzése gyakran nehézségekbe ütközik. Az elterjedt, szinte mindig beszerezhető C-64-hez készített joystick egy átalakító adapter segítségével jól illeszthető gépünkhöz. Az automata tűz ugyan általában nincs összekötve az adapter két vége között, de erre nincs is szükségünk, mert a Plus/4 gép billentyűit, így a tűzgombot is, ha lenyomva tartjuk ismétlik önmagukat.

A forgalomban levő Plus/4 gépeken a joy csatlakozók alatt eltérő jelzést láthatunk:

1./ joy 0 - joy 1
2./ joy 1 - joy 2

Ennek az eltérésnek nincs jelentősége.

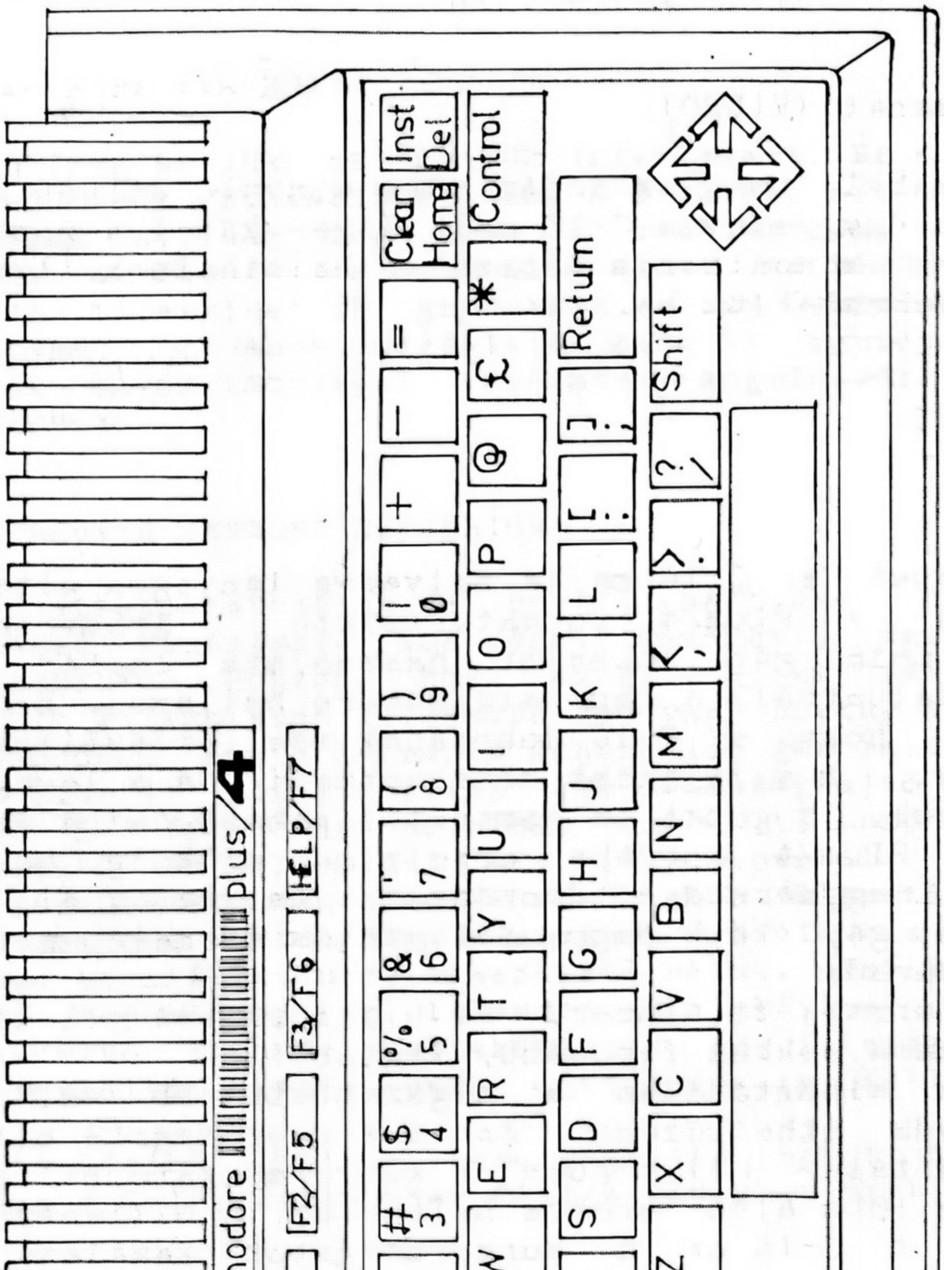
10. Videó kimenet (VIDEO)

A monitor kábel csatlakozóját ebbe a nyolc pólusú aljzatba illesszük, ugyanúgy, mint a C-16-nál. Ilyen kábel a gépnek nem tartozéka, csak a monitorok dobozában találhatunk ilyet, vagy szaküzletben szerezhetjük be.

A BILLENTYŰZET

A Plus/4 gépet a C-16 mellé helyezve lényeges eltéréseket vehetünk észre. A Plus/4 formatervezése áramvonalasabb, modernebb. A C-16 gép hosszabb használata esetén (főleg bővített memóriájúaknál) a gép melegedésre hajlamos. Ennek okai közé tartozik, hogy a C-16 dobozának szellőzőnyílásai alul helyezkednek el. A melegedést csökkentheti, ha a levegő útját szabadon hagyjuk, a gépet az asztaltól néhány cm-re megemelve használjuk. A Plus/4 esetében erre nincs szükség, mert a gép felső oldalán nagyméretű hűtőbordázatot helyeztek el, melynek köszönhetően a gép több napos folyamatos üzemelés mellett is egész langyos marad.
A billentyűk formája is újszerű, az ujj formáját jól követi. A billentyűk nyomása sokkal finomabb, lágyabb.
A billentyűk kiosztásában a legszembetűnőbb változás a funkcióbillentyűk áthelyezése, és a kurzormozgató billentyűk újszerű kialakítása. Célszerűen a kurzormozgató billentyűket egy csoportba a jobb alsó sarokba helyezték. A billentyűk nyíl alakú gombok. A C-16-on a kurzormozgatók kezelése kissé nehézkes.
A Plus/4-en a billentyűk száma eggyel megnőtt amiatt, hogy ezen a gépen két darab CONTROL billentyű található. Ezek ugyanazt a funkciót töltik be, mint a C-16-on a CTRL billentyű. Az összes billentyű tartósan lenyomva tartva önmagát ismétli.
A funkcióbillentyűk áthelyezését valószínűleg a minél kisebb méretre való törekvés indokolta. A funkcióbillentyűk jelentése csak az F1 esetében tér el a C-16-étól. Az F1 billentyű és a RETURN megnyomásának hatására rendelkezésünkre áll a négy beépített szoftver, amelyekre a gép neve is utal. E változtatás folytán a grafikus üzemmód csak képernyőre gépelt parancs segítségével kapcsolható.

A Plus/4 billentyűzete:



BASIC

A Plus/4 számitógépet bekapcsolva TV készülékünkön a következő üzenet jelenik meg:

```
COMMODORE BASIC V3.5 60671 BYTES FREE
3-PLUS-1 ON KEY F1

READY.
```

A felirat arról tájékoztat bennünket, hogy a Plus/4 gép a BASIC 3.5 verziója szerint programozható. Ez minden tekintetben megegyezik a C-16 BASIC nyelvjárásával.
A 60671 szám arra utal, hogy BASIC programjaink számára ennyi byte a szabad memóriaterület. Ez majdnem ötször annyi, mint a C-16 vagy C-116 esetében.

TARFELOSZTAS

Az alapkiépitésü C-16 memória felosztása a következő:

```
                                   R O M
                       FFFF   ++++++++++++++
                              +            +
                              +            +
                              +            +
                       C000   +            +
                              ++++++++++++++
                       BFFF   +            +
                              +            +
                              +            +
                              +            +
                       8000   ++++++++++++++

          R A M
   ++++++++++++++   3FFF
   +            +
   +            +
   +            +
   +            +
   ++++++++++++++   0000
```

A bövitett C-16 memória térképe:

```
                R A M                           R O M
        +++++++++++++++     FFFF     +++++++++++++++
        +-------------+     FD00     +             +
        +             +              +             +
        +             +              +             +
        +             +     C000     +             +
        +++++++++++++++              +++++++++++++++
        +             +     BFFF     +             +
        +             +              +             +
        +             +              +             +
        +             +     8000     +             +
        +++++++++++++++              +++++++++++++++
        +             +     7FFF
        +             +
        +             +
        +             +     4000
        +++++++++++++++
        +             +     3FFF
        +             +
        +             +
        +             +
        +++++++++++++++     0000
```

Az alapkiépitésü Plus/4 tártérképe a következö:

```
            R A M                  R O M                  R O M
FFFF   +++++++++++++++     +++++++++++++++        +++++++++++++++
FD00   +-------------+     +             +        +             +
       +             +     +             +        +             +
       +             +     +             +        +             +
C000   +             +     +             +        +             +
       +++++++++++++++     +++++++++++++++        +++++++++++++++
BFFF   +             +     +             +        +             +
       +             +     +             +        +             +
       +             +     +             +        +             +
8000   +             +     +             +        +             +
       +++++++++++++++     +++++++++++++++        +++++++++++++++
7FFF   +             +
       +             +      Rendszer ROM           Belsö  bövitö
       +             +                                  ROM
4000   +             +
       +++++++++++++++
3FFF   +             +
       +             +
       +             +
       +             +
0000   +++++++++++++++
```

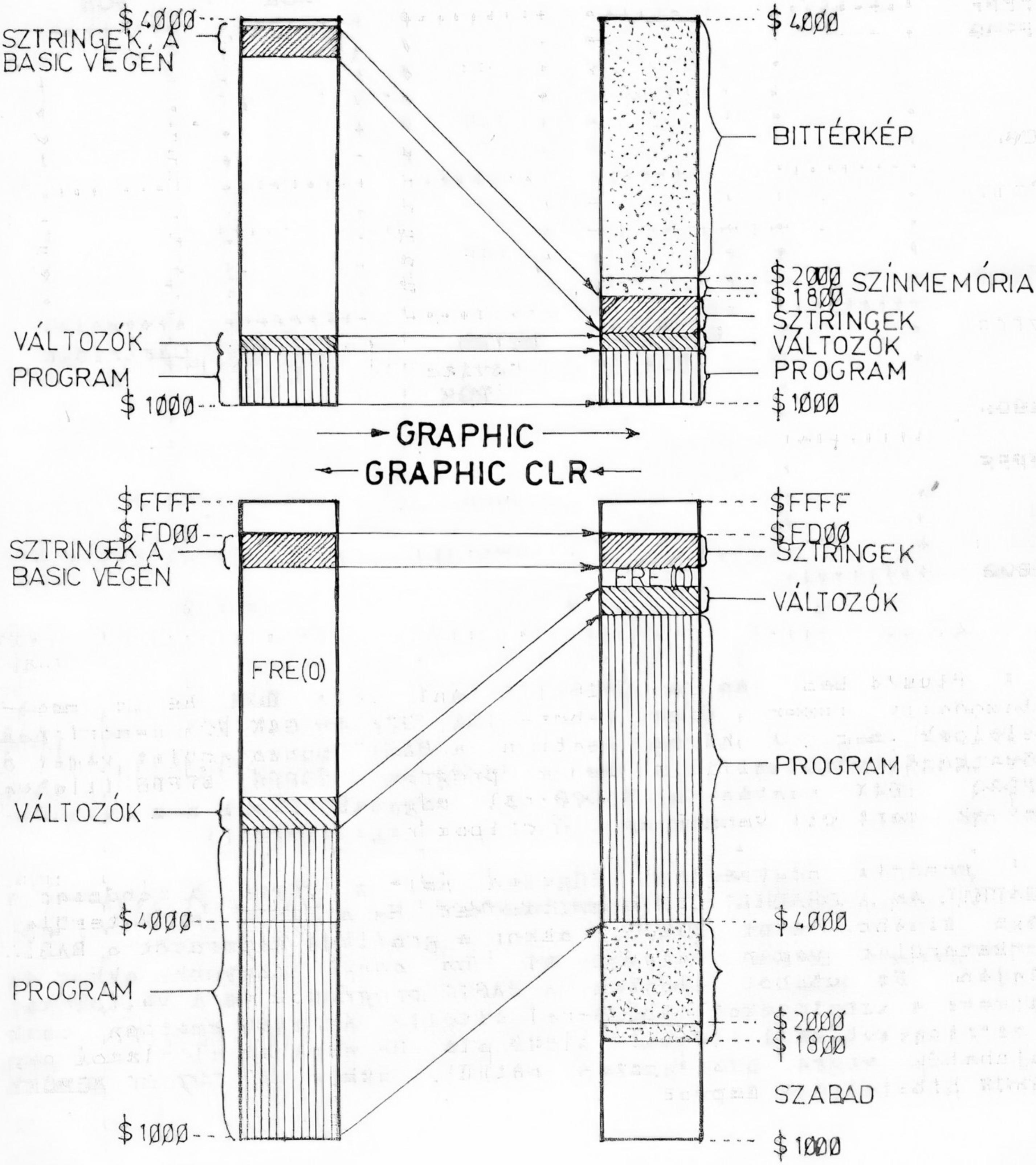
A C-16 és a Plus/4 teljes RAM-ROM kiépitése esetén a tárfelosztás a következő:

```
             RAM         ROM         ROM         ROM         ROM
FFFF    +++++++++   +++++++++   +++++++++   +++++++++   +++++++++
FD00    +-------+   +       +   +       +   +       +   +       +
        +       +   +       +   +       +   +       +   +       +
        +       +   +       +   +       +   +       +   +       +
        +       +   +       +   +       +   +       +   +       +
C000    +       +   +       +   +       +   +       +   +       +
        +++++++++   +++++++++   +++++++++   +++++++++   +++++++++
BFFF    +       +   +       +   +       +   +       +   +       +
        +       +   +       +   +       +   +       +   +       +
        +       +   +       +   +       +   +       +   +       +
8000    +       +   +       +   +       +   +       +   +       +
        +++++++++   +++++++++   +++++++++   +++++++++   +++++++++
7FFF    +       +    Rendszer     Belső     Cartridge   Cartridge
        +       +      ROM        bővítő        1           2
        +       +                  ROM
4000    +       +
        +++++++++
3FFF    +       +
        +       +
        +       +
        +       +
0000    +++++++++
```

A Plus/4-ben (és a C-16/116-ban) levő ROM három memóriamodellt ismer. Ezek rendre 16K, 32K és 64K ROM memóriának felelnek meg. A három esetben a BASIC munkaterület végét a következőképpen állitja be a program: $3FF6, $7FF6 illetve $FD00. (64K esetén a $FD00-nál magasabb cimek nem használhatóak, mert ott vannak az I/O chipek regiszterei!)

A memória nagyságától függően hajtja végre a rendszer a GRAPHIC és a GRAPHIC CLR utasitásokat. Ha a BASIC munkaterület vége kisebb, mint $4000, akkor a grafikus képernyőt a BASIC munkaterület végén helyezi el, ha ennél nagyobb, akkor az elején. Ez utóbbi esetben a BASIC programot és a változókat (kivéve a sztringeket) $3000-rel eltolja. Az első esetben csak a sztringeket kell eltolnia $1800 alá. Ha ezek az eltolások nem hajthatók végre adatvesztés nélkül, akkor ? OUT OF MEMORY ERROR hibajelzést kapunk.

A kis és nagymemóriás gépekben a GRAPHIC és GRAPHIC CLR utassítások hatására a BASIC munkaterület átrendeződése a következő:



SZOFTVERKOMPATIBILITAS

A bevezetöben már emlitettük, hogy a programok nagyobbik részére igaz, hogy mindkét (C-16, Plus/4) gépen müködnek. Gondot csupán a kibövitetlen C-16 alapgépre írt egyes programok Plus/4, ill. bövített C-16 gépeken való futása okozhat.

Ha egy kis memóriás programot a Plus/4-be vagy 64K-ra bövített C-16-ba töltve a futtatás során csak kriksz-krakszokat látunk a képernyön még nem biztos, hogy le kell törölni a programunkat, vagy szervízbe kell vinni gépünket. A hiba oka valószinüleg az, hogy programunkban a négyszeres memóriaterületen nincs egyértelmüen definiálva a karakterterület pontos kezdöcíme. Ezekben az esetekben a programot a fenti hiányosság kiküszöbölésével müködöképessé tehetjük. A karakterkészlet címét, a program inditása elött be kell állítani a $FF13-as címen, mert a program írója a C-16-os alapgép adottságait kihasználva azt egy RAM-ROM átkapcsolással állította be.

Az alábbiakban közlünk egy célszerü és egyszerü programjavítási módszert, amely általában célra vezet.

```
MONITOR <RETURN>

A 0FE8 LDA #$10
       STA $FF13
       JMP (a program eredeti kezdöcíme)
```

A program új kezdöcíme a jelen átírás után $0FE8 lesz.

A Plus/4 gépre írt programok C-16-on való futtatásakor semmilyen gondunk nem lehet, amennyiben belefér a C-16-os szük szabad memóriájába, vagy rendelkezünk megfelelö tárbövítövel.

Ha azt szeretnénk, hogy a Plus/4 úgy müködjön, mint a C-16, akkor írjuk gépünkbe ezt a kis BASIC programot.

```
10 KEY 1 ''GRAPHIC''
20 A=DEC(''37'')
30 POKE A, DEC(''F6''): POKE A+1, DEC(''3F'')
40 CLR
50 NEW
RUN
```

MITÖL PLUS/4?

A gép a 'Plus/4' elnevezést onnan kapta, hogy rendelkezik egy különlegességgel, ami abból áll, hogy egyetlen gombnyomással n é g y egymással kölcsönhatásban álló beépitett szoftver használatára nyilik lehetöségünk.
Ezek a beépitett programok a következök: szövegszerkesztö, számviteli tömb, grafikon késztö, adatbázis kezelö.

A programok részletes ismertetésére nem térünk ki, mivel ezt mindenki elolvashatja a Novotrade kiadványában, amelynek cime: A beépitett programok kezelése.

A beépitett programok közül talán a legjobban használható a szövegszerkesztö. Ennek a könyvnek a Plus/4 gépröl irt fejezete is ennek a programnak a felhasználásával készült. A szövegszerkesztö a Novotrade '3+1 magyarul' cimü programjával ékezetesithetö. Az igy lemezre kimentett szöveget nyomtatáshoz az emlitett program nélkül kell betölteni.

A szövegszékesztö lehetövé teszi, hogy a megirt szöveget mágneslemezen tárolva bármikor újragépelés nélkül javitsuk, átszerkesszük, kiegészitsük.

A programnak csak néhány szolgáltatását emeljük ki.

Egy file maximum 99 sorból és 77 karakterböl állhat. A sorokba karaktereket, a sorok közé újabb sorokat szúrhatunk be, vagy törölhetünk ki. A lapszéleket megváltoztathatjuk, tabulálhatunk a szöveg irása közben, vagy utólag. A center utasitás a szöveg egy sorát viszi a lap közepére. Véletlenül törölt szavak, sorok visszahozhatók a képernyöre. A justify utasitás hatására a szöveg úgy húzódik szét, hogy minden sor utolsó betüje a jobb oldali margóra kerüljön. Ha be van kapcsolva a szókezelö, a szavak nyomtatásnál nem lesznek szétválasztva. Szavakat, kifejezéseket lehet egy utasitással keresni, cserélni a szövegben. Lehetöségünk van szövegek összefüzésére. Be lehet állitani, hogy egy oldalra hány sort nyomtassunk. Oldalak számozhatók, nyomtatási szünet, kötelezö lapváltás elöre beállitható. Az asc utasitás lehetövé teszi az ASCII karakterek beiktatását.

A szövegszerkesztö után a legjobban használható beépitett program az adatbázis kezelö. Ez alkalmas lemezoldalanként maximum 999 rekord tárolására. Egy rekordban maximum 17 adatmezö lehet. Egy adatmezö maximális mérete 38 karakternyi. A létrehozott adatállományt listázhatjuk, vagy különbözö általunk meghatározott szempontok szerint válogathatunk az adatok között.

Készült: VASKUT MÜFIL Nyomda – KARTON GM.
88. 026.
Kiadó: LSI ATSz
Felelős kiadó: dr. Kovács Magda

Ár



Postacím:

BUDAPEST
POSTAFIÓK 121.
1300

LSI ALKALMAZÁSTECHNIKAI
TANÁCSADÓ SZOLGÁLAT